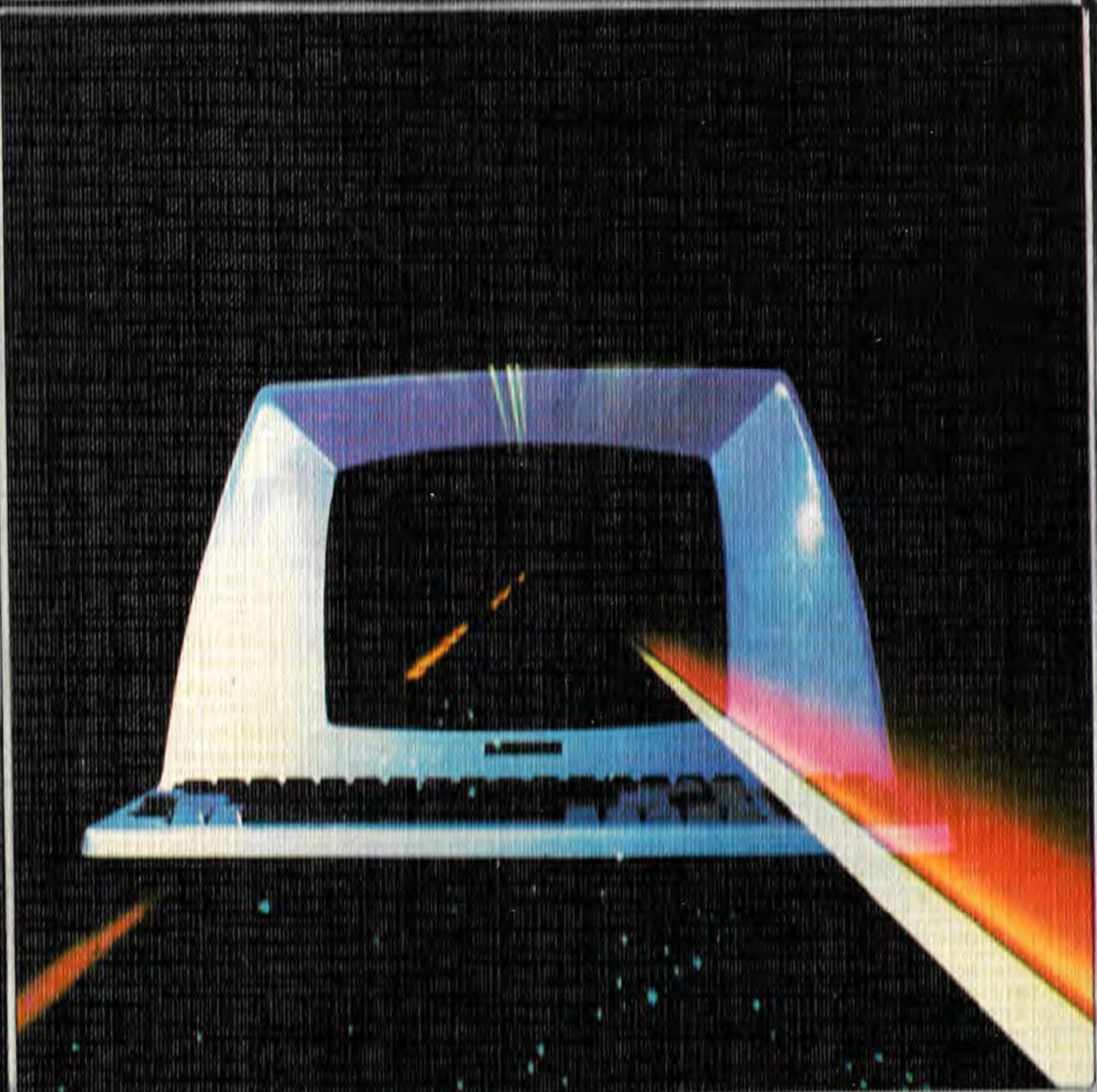
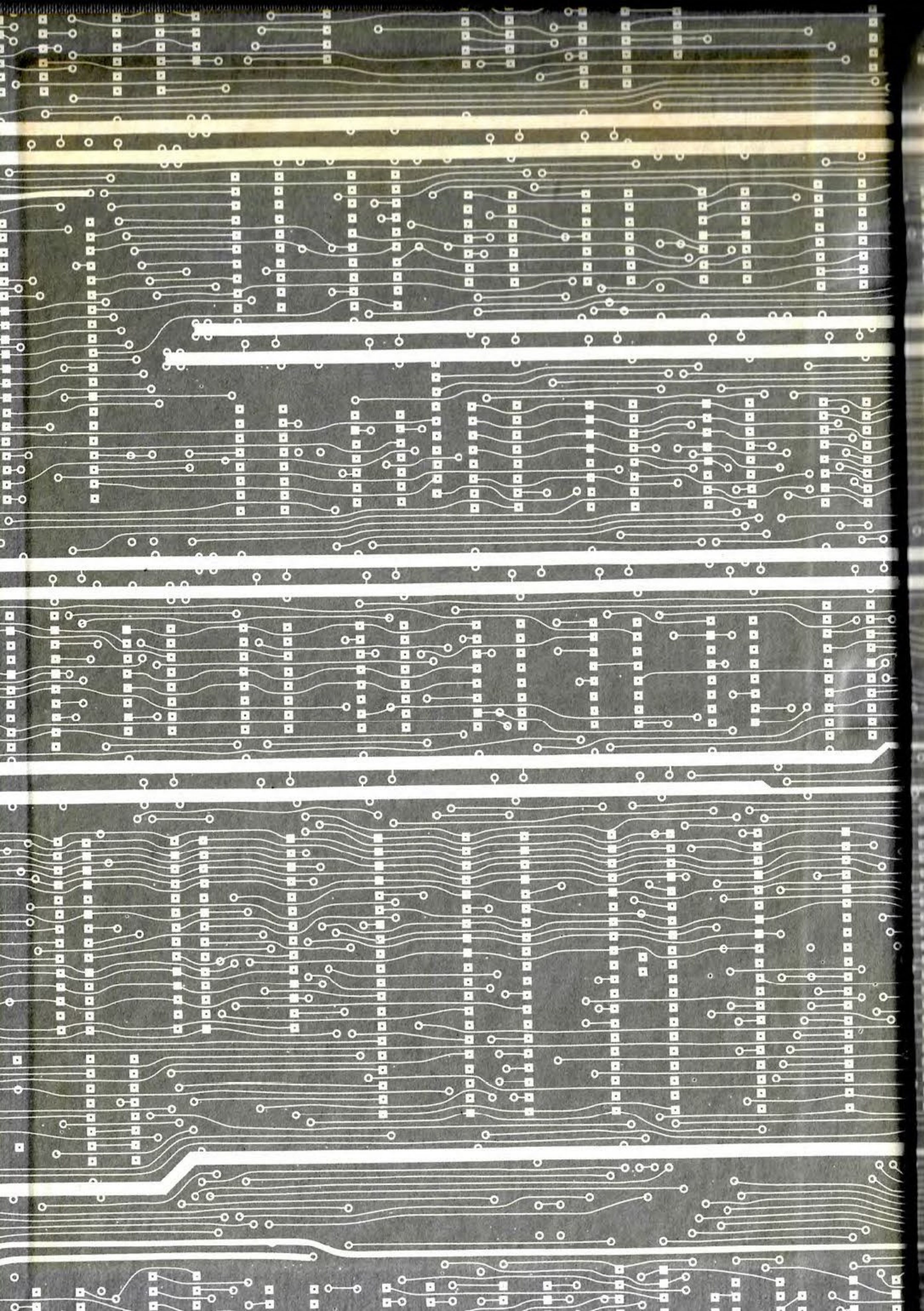
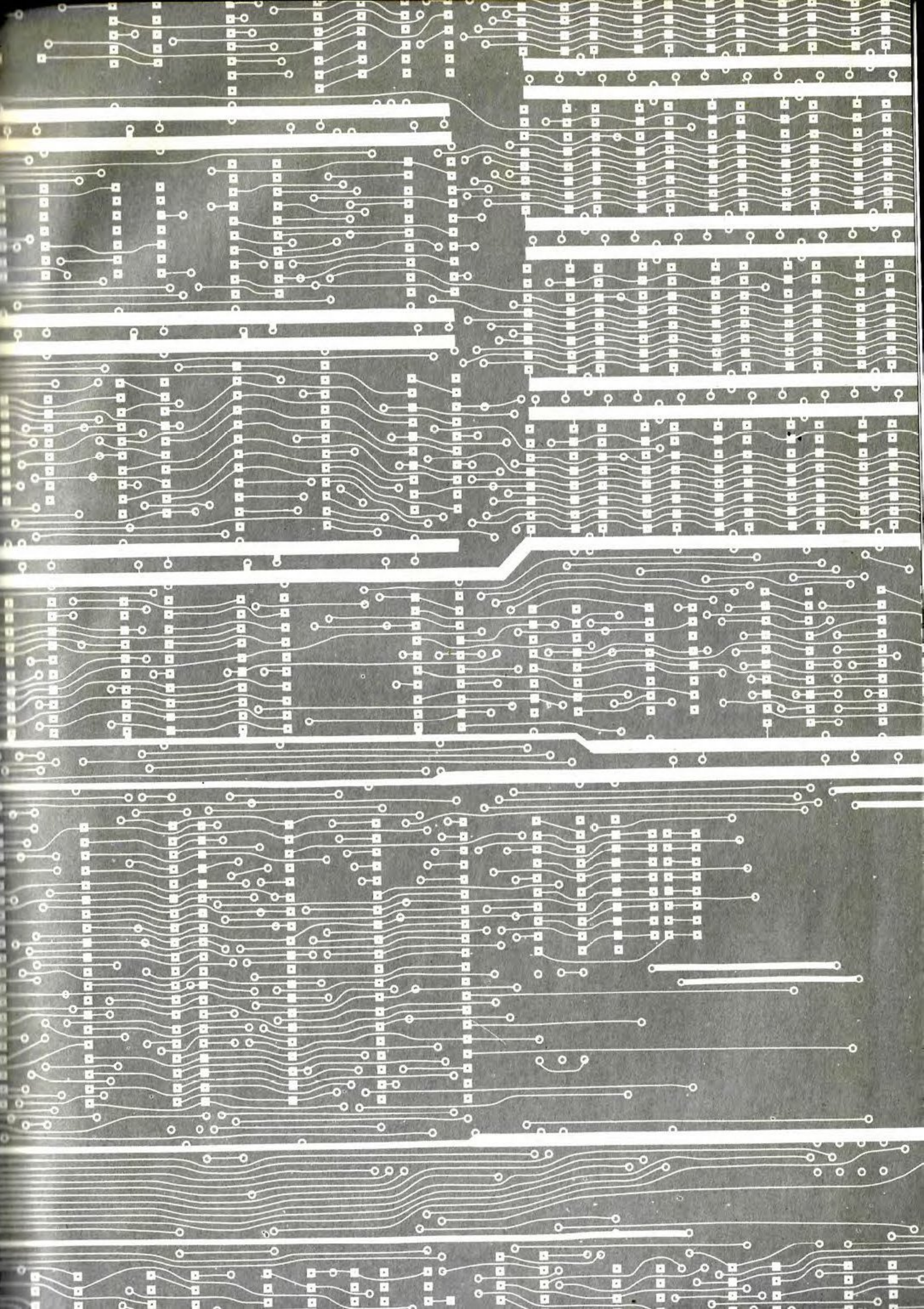
BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES



EDICIONES FORUM





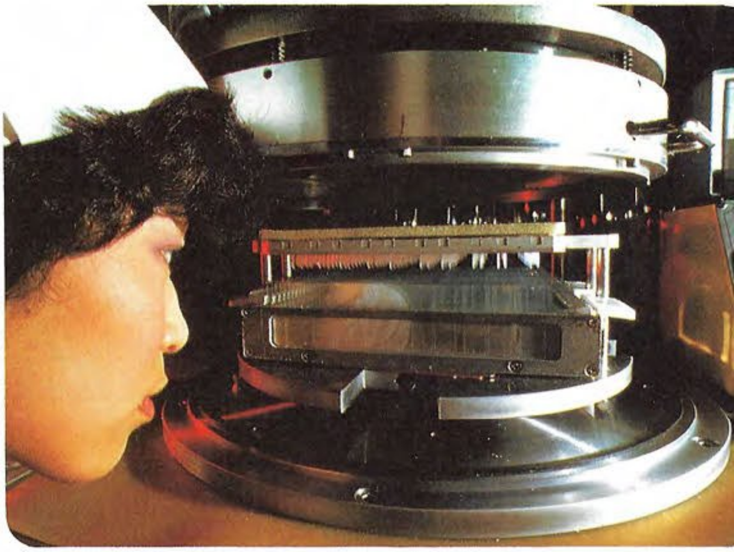
GUILLERMO

1221/51

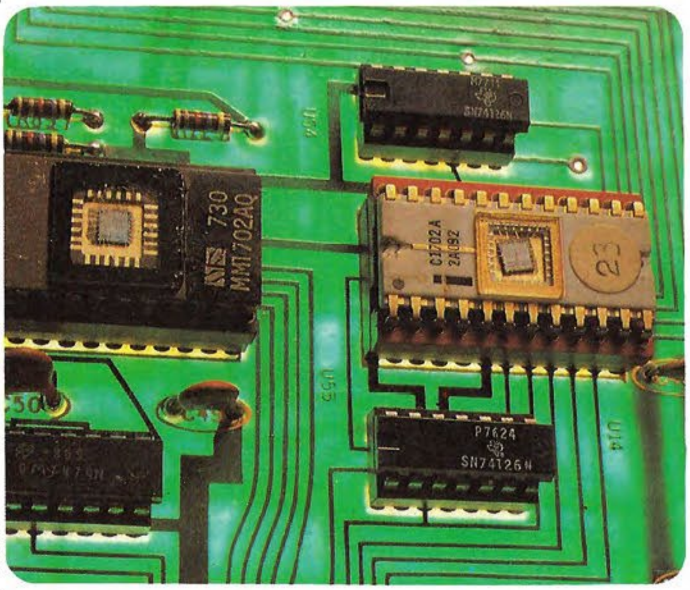
BASIC

ENCICLOPEDIA DE LA INFORMATICA.
MINIORDENADORES Y ORDENADORES PERSONALES

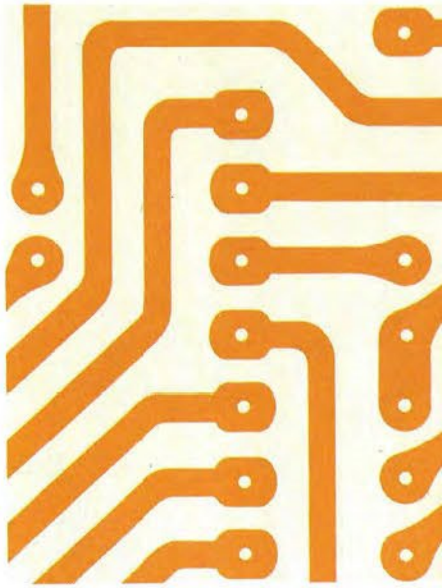
ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES



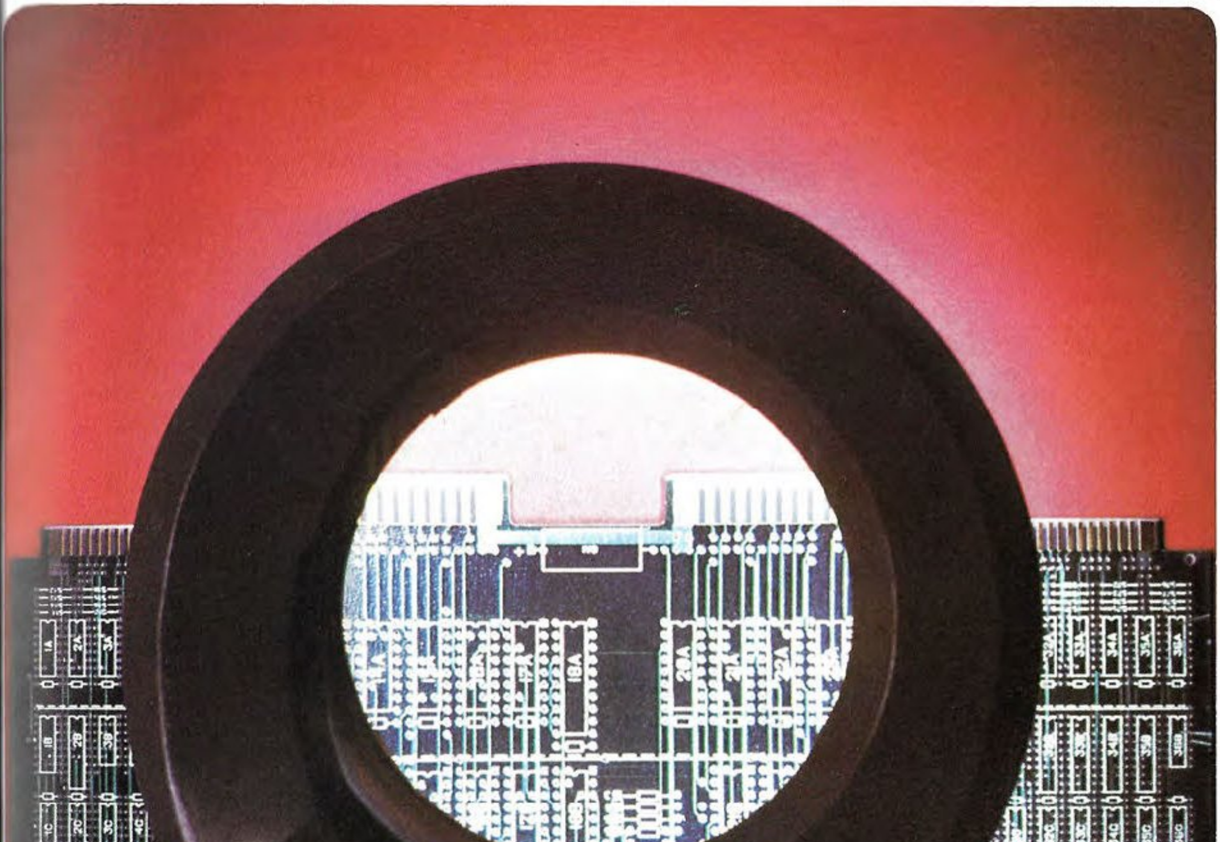
M. Wolf/Black Star-Grazia Neri



Marka



BASIC 2



BASIC

ENCICLOPEDIA DE LA INFORMATICA. MINIORDENADORES Y ORDENADORES PERSONALES

Presidente
José Manuel Lara

Director Ejecutivo
Jesús Domingo

Dirección editorial
R.B.A., Proyectos Editoriales, S.A.

Dirección técnica
Sante Senni

Programas de **Enrico Calcara**

Con el asesoramiento de la Sociedad **E.G.S.**

REDACCION

Dirección: Gabriella Costarelli
Redactor jefe: Marcella Marcaccini
Secretaría de redacción: Giulia Abriani, Maria Pierantozzi
Redacción: Ugo Spezia
Corrección: Maria Albergo, Laura Salvini, Graziella Tassi
Recopilación material gráfico: Carla Bertini, Rossella Pozza
Producción: Piergiorgio Palma
Secretaria de producción: Maria Rita Ciucci
Diseño y dirección artística: Vittorio Antinori
Jefe estudio gráfico: Roberto Sed
Compaginación: Alberto Berni, Riccardo Catani, Patrizia Fazio
Dibujos: R. Giorgini, R. Lazzarini, G. Mazzoleni, R. Mazzoni
Traducción: Carlo Frabetti
Diseño cubierta: Neslé Soulé
Jefe de Producción: Ricardo Prats

Composición electrónica: ITC-Fototipo. Barcelona-Madrid.
Imprime: CAYFOSA - Carretera de Caldas, Km. 3,7
Santa Perpetua de Mogoda (Barcelona)

Depósito Legal: B. 37.099/83
ISBN (Obra completa): 84-7574-040-5
ISBN (Volumen II): 84-7574-109-6
ISBN (Fascículos): 84-7574-044-8
Impreso en España - Printed in Spain

El editor agradece la colaboración de:
Alfa Romeo, Buffetti Data, Commodore Italiana, CPT Italia, Creazioni Walt Disney, Data General, Digital, Doxa, Elsag, Ericsson, Facit Data Products, Ferrari, FIAT, Harden Italia, Hengstler Italia, Hewlett Packard, IBM, Intema, IRET Informática, Italcable, Lancia, Litton BEI, MEE, MSI Data Italia, Olivetti, Perkin-Elmer, Plessey Trading, Prime Italia, Rank Xerox, Rhône-Poulenc Italia, Sanco Ibex Italia, Sarin, Selca Elettronica, Selenia, Sperry, SIP, Telespazio.

El planteamiento de los programas

El planteamiento del trabajo es la fase más delicada de toda la actividad de programación. En las aplicaciones con grandes sistemas (main frame), la actividad la desarrollan distintos profesionales, cada uno con una tarea propia y claramente definida.

En los microordenadores y ordenadores personales, es el mismo programador quien lleva a cabo tanto la parte relativa al planteamiento como la tarea de programación.

La lógica a seguir en la fase de planteamiento es común a todos los problemas prácticos, y se resume en unos pocos puntos principales:

1 / Definición de las salidas necesarias
2 / Comprobación de los datos de entrada y de los algoritmos de cálculo
3 / Elección de la máquina

Tras haber tomado las decisiones pertinentes en cada uno de los puntos indicados, se puede proceder a la realización de los diagramas de flujo y a la escritura de los programas.

Definición de las salidas necesarias

El primer paso consiste en establecer la clase y el volumen de las salidas que se necesitan. En esta fase, los problemas a definir son:

1 / Listados: cuántos son, qué datos han de contener y con qué frecuencia serán requeridos
2 / Files: cuáles y cuántos son los parámetros y las variables que hay que memorizar en disco

Configuración típica de un ordenador personal de mediana envergadura.



IRET Informatica

El primer punto, además de definir las variables de salida, sirve para determinar el tipo de impresora más adecuado.
Partiendo de la longitud prevista de los listados, se puede determinar la velocidad de impresión más conveniente (expresada en caracteres por segundo) y la longitud de la línea, según los tiempos requeridos en cada caso.
Huelga decir que en las aplicaciones con ordenador personal la elección de la impresora más adecuada para generar el output de un determinado programa depende de la configuración del sistema concreto del que se dispone. Una evaluación a nivel más general del tipo de tarea que pensamos realizar con nuestro ordenador personal nos permitirá elegir la impresora más adecuada en función de la oferta del mercado, las consideraciones económicas y el volumen de trabajo habitual previsto. Los principales tipos de impresora existentes hoy en el mercado del hardware se enumeran a continuación, junto con sus principales características.

Impresoras de agujas

80 columnas (es decir, imprimen 80 caracteres por línea, en hojas de reducidas dimensiones)
100 caracteres por segundo (alrededor de 1,25 líneas por segundo)

132 columnas (hojas de formato grande)
140 caracteres por segundo (1,25 líneas por segundo)

132 columnas
200 caracteres por segundo

Las impresoras de agujas suelen ser bidireccionales. La cabeza de escritura imprime los caracteres tanto en el recorrido de ida de un extremo a otro del papel, como en el de vuelta.

Impresoras de margarita

Las velocidades de impresión están comprendidas, por término medio, entre 20 y 50 caracteres por segundo.

Impresoras paralelo

Poseen velocidades de varios centenares de líneas por minuto.

Las impresoras más usadas para los microordenadores son las de agujas. Entre ellas, la elección del modelo se efectúa en base al formato del papel y la velocidad (en líneas por minuto) más idónea. Las impresoras de margarita tienen un costo casi doble que las restantes, y sólo se usan cuando se desea una alta calidad estética de los caracteres.
Las impresoras paralelo cuestan unas diez veces más que las de aguja, y no están justificadas, desde el punto de vista económico, sino en los miniordenadores más potentes. La segunda fase de la tarea de definición de las salidas requeridas permite establecer el orden de magnitud de la cantidad de datos a memorizar.
Entre los datos a elaborar a lo largo de un programa, normalmente hay algunos que tendrán que ser memorizados en disco. La suma de los espacios físicos necesarios para ello suministra el orden de magnitud del espacio ocupado por los files y, por tanto, la capacidad que han de tener los discos.
Las unidades floppy, como ya sabemos, tienen capacidades comprendidas entre 160.000 y 1.200.000 caracteres; las de los discos fijos van de 5.000.000 de caracteres en adelante.
Puesto que el costo del sistema depende casi en un 50% del tipo de disco empleado, las distintas elecciones suponen notables diferencias en lo económico.
Sin embargo, hay que tener en cuenta las futuras aplicaciones; los archivos están destinados a ampliarse con el tiempo y, por tanto, la necesidad de contar con un margen aceptable puede inducir a adoptar una solución más cara.

Comprobación de los datos de entrada y de los algoritmos de cálculo

Una vez finalizada la definición de las salidas requeridas, hay que comprobar que existan todos los datos de entrada necesarios para el ordenador y que se conozcan los eventuales métodos de cálculo. Los principales objetivos de esta fase son, por tanto, los siguientes:

- Comprobación de la disponibilidad de los datos necesarios
- Evaluación de la necesidad de memorizar algunos datos en disco
- Definición de los métodos de cálculo

Algunos de los datos de entrada pueden servir sólo como elementos de cálculo sin comparecer nunca en los listados; pese a ello, puede ser igualmente necesario memorizarlos en disco y,

por tanto, hay que tener en cuenta este factor en el momento de evaluar la capacidad de memoria masiva necesaria.

Elección de la máquina

Es la última fase del análisis, y es la que más condiciona la escritura de los programas.
Hasta ahora hemos hablado de los componentes «periféricos» del sistema, es decir, la impresora y la unidad de disco; el último paso es la elección de la unidad central.
Básicamente, hay dos tipos principales de ordenador, que difieren de forma sustancial:

- **ordenadores de 8 bits**
- **ordenadores de 16 bits**

Los del primer tipo pueden direccionar hasta 64.000 memorias, mientras que los del segundo pueden llegar, en algunos casos, al millón.
La elección de uno u otro tipo depende, por tanto, de la cantidad de datos a elaborar. En los ordenadores de 8 bits, si la masa de datos es importante, hay que segmentar los programas, lo que aumenta las dificultades de programación. Sin embargo, estos sistemas tienen un costo menor. Los ordenadores de 16 bits tienen un costo inicial mayor y ulteriores aumentos de costo para cada incremento de memoria. Además, aunque la capacidad de direccionamiento sea muy elevada, algunos sistemas operativos no pueden aprovecharla plenamente; el usuario tiene, pues, una máquina potencialmente capaz de direccionar, por ejemplo, 1 Mbyte, pero el sistema operativo sólo puede tener acceso a un área mucho más restringida.
El último componente en la elección de la máquina es precisamente el sistema operativo. La elección más conveniente es la que se orienta hacia las máquinas dotadas de compiladores y que utilizan sistemas operativos de amplia difusión; así se tiene la ventaja de la trasponibilidad de los programas y la seguridad de poder disponer de un sistema operativo que, dada su difusión, será actualizado continuamente.

Sistemas operativos

Como hemos visto, el ordenador, en su forma más simple, es un aparato electrónico capaz de recibir y elaborar señales eléctricas de tipo digital binario. Podemos pedirle que opere de una determinada manera con unos datos que se le suministran en la entrada de forma binaria, junto con las necesarias indicaciones sobre la serie de funciones a realizar. La serie ordenada de los códigos de las distintas funciones a realizar constituye el programa.
Para programar una máquina de este tipo habría que utilizar necesariamente la simbología binaria, escribiendo datos y códigos directamente en lenguaje máquina. Cada una de las funciones a realizar por la máquina, hasta la más banal, tendría que ser «descrita» de esta forma en cada programa.
Es fácil comprender que la gestión de un hardware de este tipo exigiría tiempos prohibitivos para la simple elaboración de los programas, y requeriría capacidades poco comunes por parte del programador. Sin embargo, las situaciones de este tipo estaban a la orden del día hace no más de veinticinco años; los programadores de entonces se veían obligados a redactar programas consistentes en series interminables de símbolos 1 y 0, y las probabilidades de cometer errores eran elevadísimas.
Para facilitar el trabajo de programación se han creado sucesivamente los lenguajes simbólicos de bajo nivel (Assembler) y de alto nivel (Basic, Fortran, Cobol, etc.).
Para eliminar además la necesidad de reprogramar cada vez las funciones de uso generalizado, existen actualmente módulos especiales (módulos de servicio) que pueden utilizarse en todos los programas de aplicación.
El **sistema operativo** es el conjunto de los módulos de servicio y de los compiladores (o intérpretes) destinados a la traducción (compilación o interpretación) de los programas simbólicos.
Veamos ahora detalladamente cuáles son las principales funciones de un sistema operativo para microordenador y ordenador personal.
Ante todo, hay que distinguir entre los sistemas operativos a utilizar en máquinas «dedicadas», que generalmente tienen un solo usuario (microordenadores), y los sistemas operativos que permiten la realización simultánea de varios programas (miniordenadores y main frame).
En el segundo caso, las funciones son más complejas, puesto que el sistema operativo ha de ser capaz de «repartir» los recursos disponibles (memorias, discos, periféricos, etc.) entre los diversos programas que elabora simultáneamente, y todo ello con una velocidad tal que, dentro de ciertos límites, cada usuario no advierta la presencia de los demás.

Las funciones del sistema operativo

Para centrar ideas, consideramos un sistema de microprocesador de categoría micro con las siguientes características (típicas de una máquina de potencia intermedia):

— un solo usuario
— memoria central de 32.000 bytes
— unidad video de 80 columnas × 24 líneas
— dos unidades floppy
— un teclado

Supongamos que esté prevista la programación en lenguaje Basic. El primer requisito del sistema operativo será, obviamente, que pueda traducir el lenguaje Basic.

En función de esta característica ya se puede elegir entre dos clases de sistemas operativos, puesto que algunos sólo incluyen intérpretes, mientras que otros tienen también el compilador. Los sistemas dotados de compilador ofrecen una mayor velocidad en la ejecución de los programas. Analizando la configuración de la que partimos, se deducen las otras funciones que hay que prever en el sistema operativo. El video, el teclado y la impresora son dispositivos de entrada y salida de datos. Se consigue que los programas de aplicación puedan requerir su uso sólo especificando cuál es el periférico implicado y cuáles son los datos a leer o escribir; el sistema operativo tendrá que intervenir para gestionar las complejas funciones que activan el hardware, es decir, los mecanismos destinados a la impresión y reconocimiento de las teclas pulsadas en la consola.

Análogamente, la utilización de los diskettes requiere una serie de funciones (por ejemplo, el posicionado de la cabeza en la zona del disco donde residen los datos) que han de ser controladas por los módulos del sistema operativo.

Por último, hay que disponer de un programa que pueda interpretar instrucciones concretas introducidas con el teclado, por ejemplo, la orden de inicio o interrupción de la ejecución de un programa. Todas estas funciones, y otras accesorias, determinan la estructura del sistema operativo. En general, un sistema operativo se puede dividir en tres partes fundamentales:

1 / Módulo de proceso de los comandos
2 / Sistema de gestión de las funciones I/O
3 / Módulo de gestión de unidad de disco

Ahora expondremos las funciones propias de cada parte y los métodos para su utilización.

El módulo de proceso de comandos es el primero en activarse al conectar la máquina.

Como ya se ha dicho, la configuración típica de un microordenador comprende dos unidades de disco. La primera, normalmente individualizada con la letra A o el número 0, ha de contener el diskette del sistema operativo. Al conectar la máquina, se activa un programa especial, que reside en la memoria, denominado **bootstrap** o **cebador.**

El cebador tiene como única función la de leer el sistema operativo en el diskette de sistema (unidad A o unidad 0) y de transferirlo a la memoria. Sólo tras esta transferencia, la máquina puede interpretar y ejecutar las instrucciones. En ese momento el ordenador informa al usuario de que está listo, normalmente con una tabla que enumera las principales características del hardware* o con un símbolo, denominado **prompt**, que aparece para indicar que el sistema está a la espera de instrucciones. En las fotografías de las págs. 293 y 294 se ven pantallas pertenecientes a dos tipos de máquina diferentes; en ambos casos, el prompt «>» que aparece junto al cursor indica que se puede comenzar la tarea.

En estas condiciones sólo se pueden dar instrucciones inherentes al sistema operativo, es decir, que cumplan funciones de base, aparte del tipo de lenguaje que se vaya a usar. Entonces el usuario tiene dos posibilidades:

— pedir la realización de funciones de base
— introducir un programa en lenguaje simbólico

La selección se efectúa introduciendo la función deseada. Por ejemplo, si se desea escribir un programa en Basic, en la segunda máquina (M20 Olivetti) hay que introducir la instrucción BA; en la primera (B 2 Buffetti Data) la instrucción a insertar es MBASIC.

El módulo del sistema operativo destinado a la interpretación de las órdenes reconoce la instrucción como una demanda de pasar a Basic, carga el intérprete (el Basic, normalmente, co-

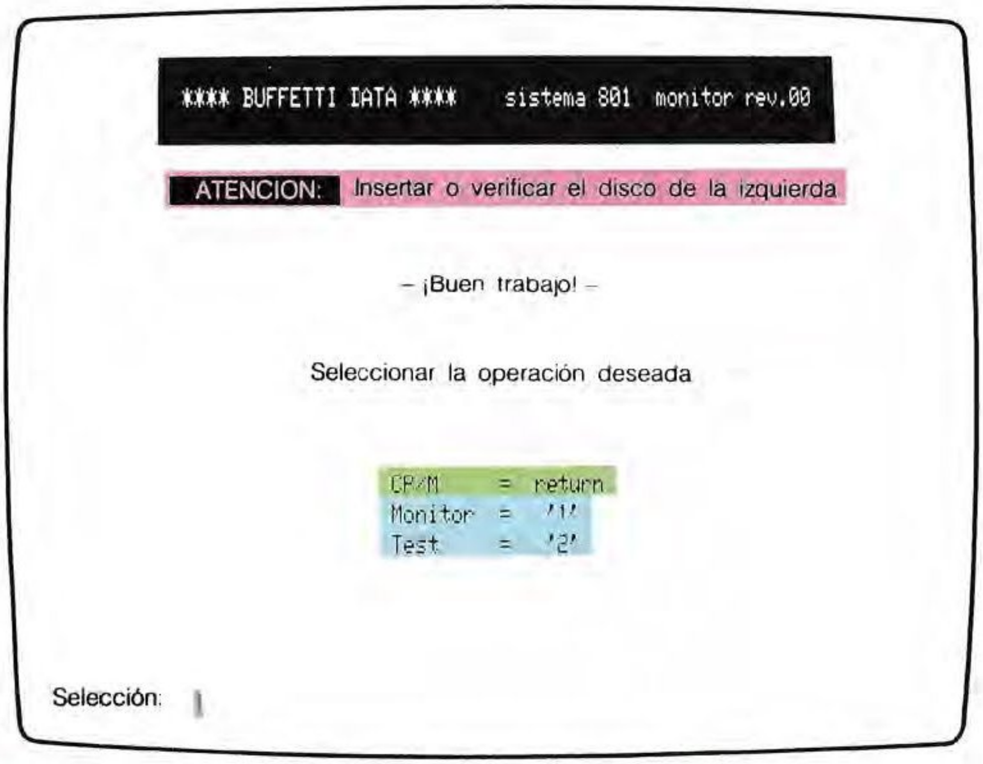
---

* Las características del hardware, es decir, la capacidad de memoria, el número de unidades de disco, etc., constituyen lo que se denomina la «configuración del sistema». La tabla que resume las características de la máquina típica a la que se hace referencia en este capítulo, es un ejemplo de configuración hardware. Un determinado sistema puede tener también una «configuración software», expresión que designa al conjunto de los programas a disposición del usuario.

VIDEO DEL SISTEMA BUFFETTI DATA 801 EN EL MOMENTO DE LA PUESTA EN MARCHA

El Buffetti Data 801 adopta el sistema operativo CP/M, que reside en disco. Para acceder al mismo, debe cargarse en memoria. En el momento de la puesta en marcha se deja expresamente abierta la unidad de disco A (la de la izquierda), en la que normalmente está inserto el diskette del SO.





El programa de inicialización, que reside permanentemente en la ROM, intenta leer los módulos del CP/M, pero encuentra que la unidad de disco no está predispuesta. Por ello, se emite un mensaje de diagnóstico.
Obsérvese que este último hace referencia expresa al disco de la izquierda (unidad A), porque en él debe encontrar el SO.
En algunos sistemas existe la posibilidad de invertir las dos unidades de disco (modificando una conexión eléctrica), lo cual permite cargar el SO también en el caso de avería de la unidad A.

El ordenador propone sucesivamente al usuario tres posibilidades diferentes de elección.

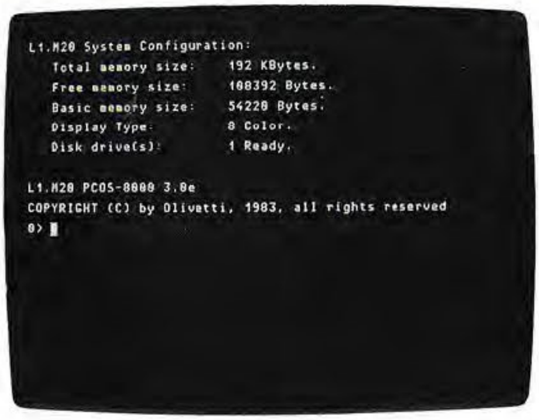
La primera permite activar las funciones del SO (comandos, Basic, etc.) pulsando la tecla RETURN.

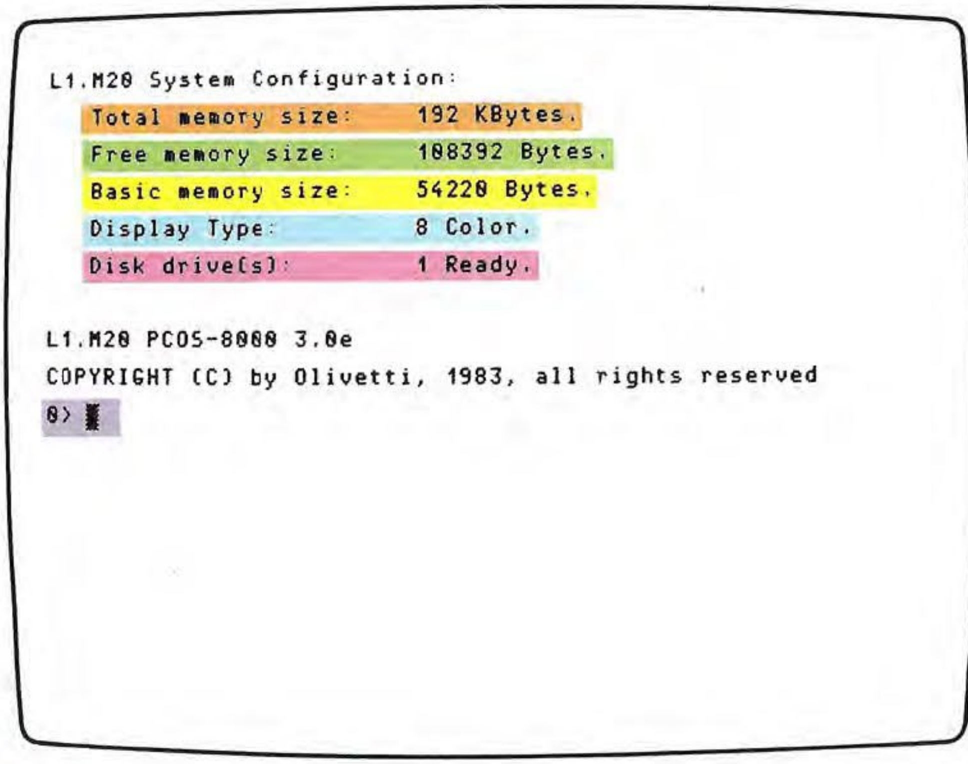
La segunda y la tercera permiten activar dos funciones auxiliares de verificación en caso de funcionamiento defectuoso.

En la última fila se presenta la solicitud de selección y aparece el cursor.

VIDEO DEL SISTEMA M20 OLIVETTI EN EL MOMENTO DE LA PUESTA EN MARCHA

Las primeras seis filas informan al usuario sobre la configuración del hardware de que dispone. En particular:





La memoria total del sistema es de 192 kbytes.

La memoria a disposición del usuario es de 108.392 bytes.

La memoria reservada a la utilización de los programas Basic es de 54.220 bytes.

El video es de 8 colores.

En línea hay disponible una sola unidad de disco. Esta señalización no excluye que el sistema también esté dotado con una segunda unidad de disco, sino que especifica que es operativa una sola de las dos. El M20 incluye dos unidades de disco integradas en vídeo, pero en el momento de tomar la fotografía, una de ellas (la número 1) estaba abierta.

Las dos filas que vienen a continuación especifican la versión y la fecha del sistema operativo que hay en línea (PCOS).

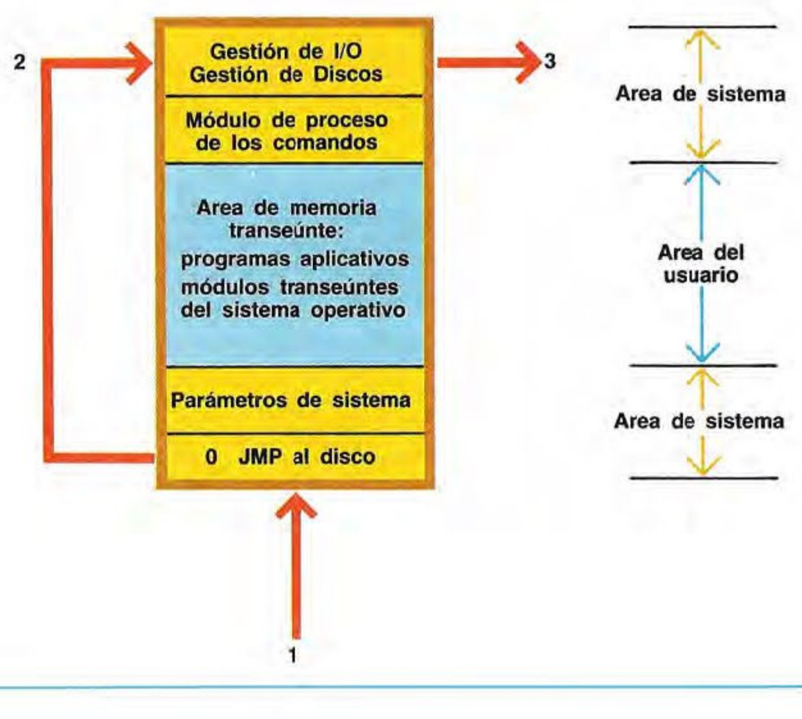
En la última línea puede verse el prompt al lado del cursor. A partir de esta posición pueden introducirse los comandos para el sistema operativo.

mienza con la forma interpretada; sólo en una segunda etapa se pasa a la compilación) y se dispone a aceptar las líneas de programa.
En el otro caso (realización de funciones de base), la instrucción a introducir está constituida por el nombre simbólico de la función deseada y por los eventuales parámetros necesarios. También en este caso el módulo de proceso de comandos analiza la respuesta suministrada y activa la función pedida.
Las instrucciones (y los respectivos programas de gestión) pueden ser de dos tipos: residentes y transeúntes.
Los **programas residentes** se hallan siempre presentes en la memoria (tras la carga del sistema operativo) y pueden ser puestos en ejecución inmediatamente; por el contrario, los **programas transeúntes** se cargan en memoria sólo cuando se requiere. El área de memoria reservada para estos programas se denomina **área de memoria transeúnte**.
El área de memoria transeúnte es también la zona en la que el usuario carga los programas de aplicación antes de la ejecución. En el gráfico inferior se muestra un ejemplo de reparto del área de memoria. La primera posición de memoria, indicada con el símbolo 0, contiene una instrucción de salto al módulo de gobierno del disco (JMP al disco), que permite la carga automática del sistema operativo. Al conectar la máquina, el contador de programa se pone automáticamente a cero, luego la primera instrucción ejecutada será la contenida en la posición 0 (punto **1**). En esta posición se halla la instrucción de salto (y activación) al módulo de carga del sistema (gestión de discos, punto **2**). Tras la carga, la máquina se pone en espera de instrucciones (punto **3**). En el mapa de memoria aparece también una zona reservada a los parámetros del sistema; en ella se memorizan todos los datos necesarios para el sistema operativo (punteros, direcciones, etc.).

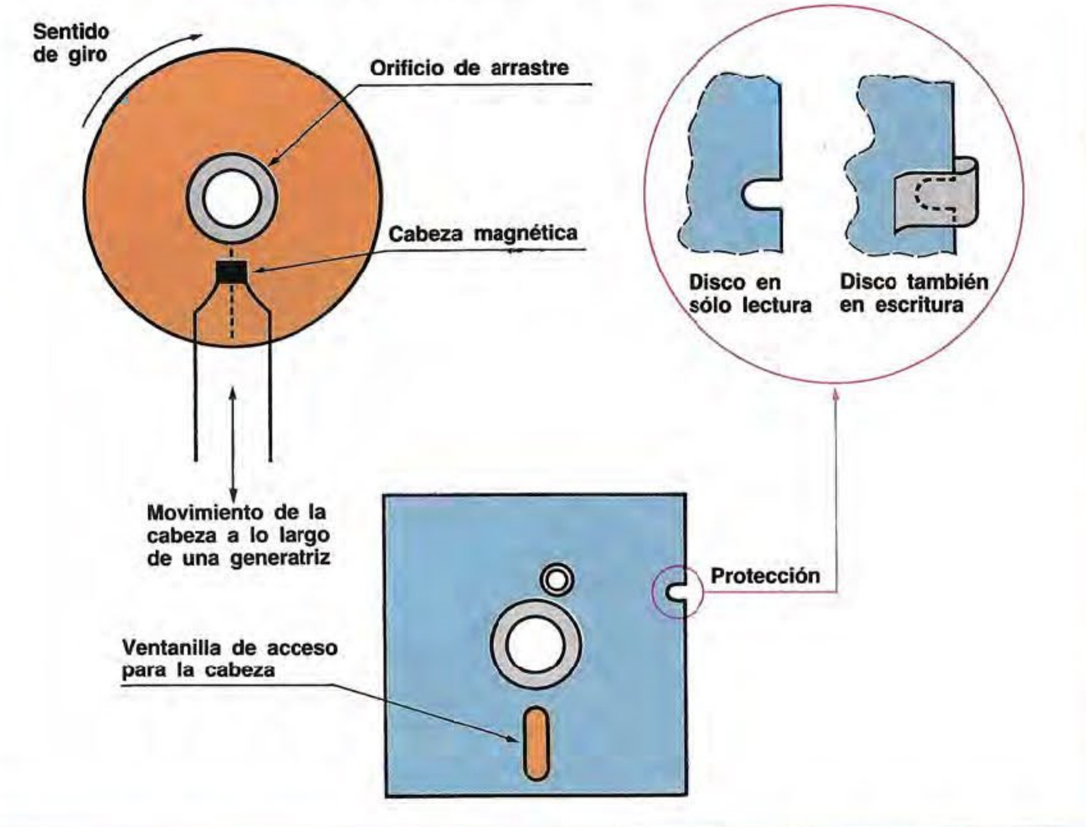
Gestión de files en floppy

Antes de comenzar el análisis de las funciones realizadas por un sistema operativo, hay que volver a ver con detalle cuánto hemos aprendido sobre la estructura física y lógica de los flop-

EJEMPLO DE DISTRIBUCION DEL AREA DE MEMORIA



ESTRUCTURA FISICA DE UN FLOPPY

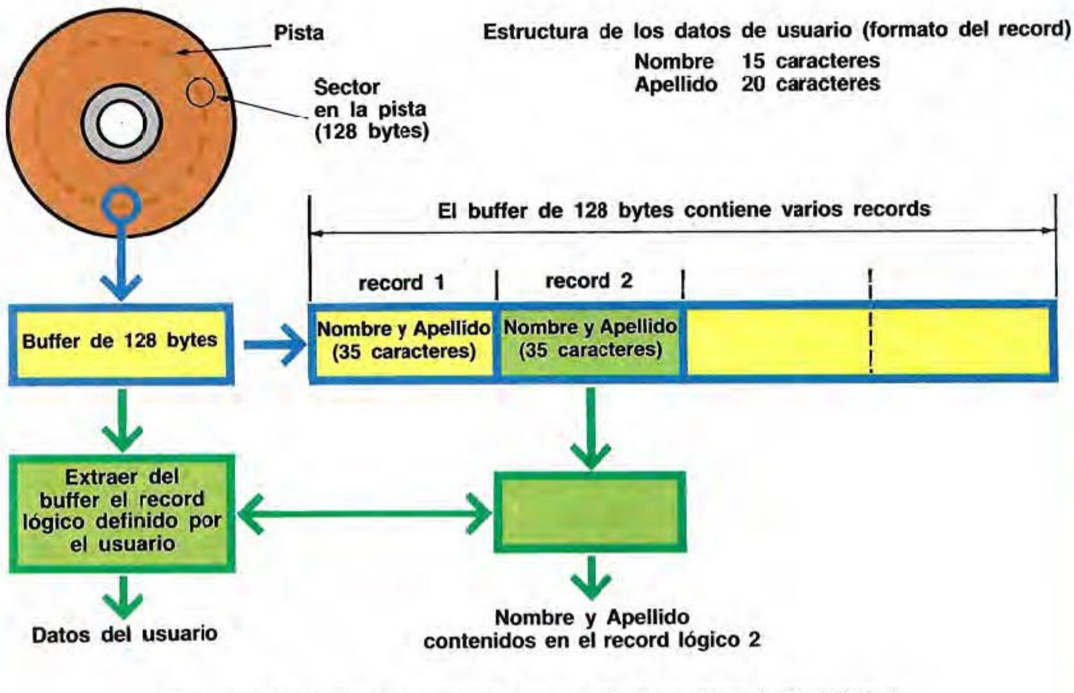


py o diskettes. Los diskettes están constituidos por un soporte flexible que está recubierto por una capa de material magnético; la presencia de magnetización indica el símbolo 1; su ausencia indica el 0.
La escritura y la lectura se efectúan mediante una cabeza magnética que puede deslizarse a lo largo de un radio del disco; simultáneamente el disco gira, y de este modo todos los puntos de su superficie pasan bajo la cabeza. Los discos están contenidos en una envoltura de protección con una ventanilla que permite el acceso de la cabeza (ver gráfico superior). En la envoltura hay una «muesca de protección»; si ésta se deja descubierta, el disco puede ser leído pero no escrito; por el contrario, si la muesca está tapada con una etiqueta adhesiva, también se puede escribir (en algunos sistemas la lógica de protección es exactamente la inversa).
Los diskettes pueden ser grabados por una sola cara o por ambas, y con zonas de magnetización más o menos próximas; la magnetización más tupida se denomina «de doble densidad» (normalmente se indica como 2D), y la menos densa «de densidad simple» (1D). La cantidad de datos (bytes) que el diskette puede contener varía desde un mínimo por una sola cara y densidad simple, hasta un máximo para doble densidad por las dos caras.

Formateado. Un disco nuevo no puede utilizarse directamente; antes hay que efectuar **el formateado**, que es tarea del sistema operativo. En esta fase el diskette es dividido en zonas denominadas «pistas», cada una de las cuales se divide en «sectores». Se establece, así, un «mapa» del área disponible, y cada zona física del diskette puede ser direccionada especificando las coordenadas correspondientes a la pista y el sector. La dirección mínima que se puede dar es la del sector; no se pueden direccionar partes del sector mismo, por lo que la escritura y lectura de datos se efectúa por bloques cuya longitud mínima es un sector.
Cada sector puede contener, normalmente, 128 o 256 bytes, y la transferencia entre disco y me-

moria tiene siempre por objeto estas cantidades de caracteres. En una segunda etapa, del total de los caracteres se toman los realmente pedidos por el programa de aplicación (ver gráfico). La estructura del diskette, es decir, el número de pistas en que está dividido, el número de sectores por pista y el número de bytes por sector, dependen del tipo al que pertenece (diámetro, simple o doble densidad, una o dos caras) y

MECANISMO DE LECTURA DE UN RECORD LOGICO



El sistema toma un sector entero y extrae de él los datos (record lógico)

del tipo de formatación que efectúa el sistema operativo. Así, son valores típicos:

- 2 caras, indicadas con 0 y 1
- 35 pistas por cara
- 16 sectores por pista
- 128 bytes por sector (1D)

La numeración de los sectores no es consecutiva sino intercalada («interleaved»), en el sentido de que el orden físico de sucesión de los sectores en una pista es 1.er sector, 3.er sector, etc.; los sectores pares están situados en la pista siguiente. Este método de grabación se adopta para minimizar los tiempos de acceso a los da-

tos. La forma más usual es la subdivisión en pares e impares, pero en algunos sistemas se puede cambiar el orden a voluntad. Por ejemplo, entre dos sectores consecutivos se pueden saltar cinco números (1.er sector, 7.º sector, etc.). Esta técnica puede llevar, en determinadas condiciones, a una ulterior disminución de los tiempos de acceso.

Directorio. Una vez efectuado el formateado, el diskette está listo para ser utilizado como memoria de masa. La memorización de datos (o de programas) tiene lugar en zonas contiguas (ficheros), cada una identificada por un nombre. Si se desea leer los datos memorizados hay que especificar en qué file se hallan: el sistema se encarga de localizar la zona física en que reside el file y de transferir su contenido a la memoria. Los programas del sistema operativo destinados a estas funciones han de conocer la posición física del file en cuestión (pista y sector de comienzo y pista y sector de final), por lo tanto para cada file memorizan nombre, dirección y clase de contenido (datos, programa). Esta zona reservada del disco es el **directorio** (file directory): en él se escriben, en el momento de su creación, los nombres de los files, su posición física y su tipo. El sistema operativo lee el directorio, obtiene la dirección del file y posiciona la cabeza de lectura en el punto correspondiente al comienzo del file deseado. En esta zona inicial del file reside un primer grupo de valores, no pertenecientes a los datos del file que se denomina File Descriptor Block (FDB) y contiene sus parámetros característicos. Por los datos contenidos en el FDB, el sistema operativo puede interpretar lo que hay escrito en el file.
Los parámetros normalmente memorizados en el área FDB son los siguientes:

- Longitud actual del file
- Número de extensiones (extent)
- Palabra de instrucción para el acceso al file (password)
- Flag de protección
- Tabla de los extent
- Puntero a la eventual extensión de la tabla de los extent

En algunos sistemas los datos del área FDB pueden ser marginalmente distintos, pero los parámetros fundamentales son comunes.
A título de ejemplo, las principales denominaciones del sistema operativo CP/M son las siguientes:

- Nombre del file (8 caracteres)
- Tipo del file (3 caracteres)
- Extent
- Amplitud del extent
- Mapa de situación del disco
- Número del siguiente record en lectura y escritura.

Como puede observarse esta tabla es similar a la tabla anterior, correspondiente al sistema PCOS (Olivetti).

Tabla de los extent. El espacio ocupado por el file no puede ser constante. Tanto si es un fichero de datos como un file de programa, a cada nueva introducción aumenta su longitud. Los records añadidos no se yuxtaponen físicamente a los preexistentes, puesto que el espacio del disco está ocupado por otros files; por tanto, hay que memorizar las ampliaciones de datos en zonas aparte y hacer una tabla que contenga los punteros a dichas zonas (las direcciones). Con la utilización de la tabla de los extent, el file, que físicamente está «troceado» en varias zonas del disco, tiene continuidad lógica. Todas las funciones de lectura de la tabla de los extent, de salto a la dirección contenida en ella y de carga de los datos en la memoria, forman parte del sistema operativo.
Un file con varias extensiones requiere, para ser leído, más operaciones que un file continuo; por esta razón es útil copiar el contenido de un diskette que contiene files discontinuos en otro nuevo. Durante la operación de transferencia, todos los files son compactados y se vuelven físicamente continuos. De esta forma se obtiene un notable ahorro de tiempo en elaboraciones sucesivas. La operación de copia del contenido de un diskette se denomina «back-up».

Referencia a los files. En los discos pueden memorizarse distintos tipos de files; los principales son:

1 / Fuentes en Assembler (ASM)
2 / Files en código máquina con formato hexadecimal (HEX)
3 / Fuentes en Basic (BAS)
4 / Fuentes en Fortran (FOR)
5 / Files que contienen programas compilados (COM)

6 / Files especiales creados por los módulos de servicio
7 / Files de datos

Los tipos del 1 al 5 van acompañados de unas siglas de tres caracteres que normalmente los designa; los otros dos tipos tienen siglas de identificación que dependen del sistema operativo y que pueden variar considerablemente de un sistema a otro.
Para hacer referencia a un file hay que dar su nombre y su tipo. Por ejemplo, si se desea cargar un programa en Basic memorizado en un file denominado PRUEBA, la referencia completa a citar es PRUEBA. BAS; la primera parte suministra el nombre del file (normalmente, 8 caracteres como máximo), y la segunda, el tipo (BAS = Basic, fuente). En algunos sistemas operativos se puede utilizar, en lugar de uno de los dos parámetros (nombre y tipo), un indicador genérico representado por un símbolo (por ejemplo *); en este caso no se da la condición restrictiva del nombre o del tipo y son aludidos todos los files.
Por ejemplo, la referencia *. BAS alude a todos los files de tipo Basic, independientemente de cuál sea su nombre.
Este método es útil en las fases de back-up. Si hay que copiar un determinado número de files fuente en Basic, conviene introducir el símbolo genérico *. BAS. De este modo son copiados todos los files de tipo BAS, cualquiera que sea su nombre. Si se especificara también el nombre, habría que introducir una instrucción para cada uno de los files a copiar.
Análogamente, introduciendo *.* se toman todos los files independientemente de su nombre y de su tipo, puesto que ambos han sido sustituidos por el símbolo genérico.

Funciones del SO relacionadas con los files

El sistema operativo contiene algunos módulos, denominados **primitivas**, destinados a la ejecución de las funciones fundamentales de gestión de los files. Cada sistema operativo tiene su propia simbología, por lo que la misma función tiene nombres distintos.
Seguidamente se da la nomenclatura más usada, o que más se aproxima a las instrucciones Basic.
Al leer este apartado hay que tener en cuenta que el acceso a las primitivas no es activado directamente por el usuario (a no ser que se utili-



La animación con ordenador

Hasta no hace mucho, la realización de una película de dibujos animados exigía muchísimo tiempo y esfuerzo. La granja de los animales, *por ejemplo, uno de los primeros largometrajes de dibujos animados, requirió 250.000 dibujos y 300.000 horas de trabajo. Pero hoy día el ordenador está alterando la fisonomía de la animación, y no sólo acelera su proceso de producción, sino que además permite al animador crear efectos completamente nuevos.*

Una cámara cinematográfica filma 24 fotogramas por segundo, produciendo una secuencia de fotografías ligeramente diferentes entre sí. Al ser proyectadas a la misma velocidad, las imágenes crean la impresión del movimiento original. En el cine de animación, sin embargo, se fotografía una serie de dibujos, uno tras otro, fotograma a fotograma.

Durante las últimas décadas, las técnicas de animación han permanecido sustancialmente iguales y se han aplicado casi por completo a mano; en la actualidad, sin embargo, se hallan en curso de perfeccionamiento sistemas basados en los ordenadores con el objeto de acelerar el laborioso procedimiento que requería dibujar 24 imágenes separadas por cada segundo de proyección.

Aunque la cámara siga utilizándose de la forma tradicional, es decir, con dibujos hechos a mano en vez de videoimágenes generadas por ordenador, el procedimiento puede igualmente acelerarse con el empleo de controles electrónicos. En este caso se utiliza una cámara instalada directamente encima de una mesa horizontal. La cámara puede subirse o bajarse con una serie de pequeños saltos, mientras que la mesa puede desplazarse hacia adelante o hacia atrás, o bien hacer que ruede.

Normalmente, el operador utiliza una determinada secuencia de movimientos de la mesa o de la cámara para obtener efectos panorámicos, para modificar el tamaño de las figuras o para producir efectos de movimiento. Por ejemplo, un personaje puede ser fotografiado en una sucesión de actitudes de carrera. Pero, para aumentar el realismo de la secuencia, el escenario de fondo puede ser desplazado hacia atrás a saltos perfectamente calculados de forma que el personaje parezca moverse hacia adelante. Si el aparato es accionado por un ordenador, se puede obtener una secuencia de movimientos mucho más precisa; se ahorra mucho trabajo y toda la acción parece más real. Cada imagen, naturalmente, ha de ser traducida al lenguaje del ordenador para que pueda elaborarla y almacenarla.

Para codificar una imagen según el lenguaje del ordenador, se divide en un mosaico de diminutos cuadraditos de color. A cada cuadradito o «pixel» (elemento de imagen) se asigna un número que representa su nivel tonal. El número podría variar, por ejemplo, del 0 para el blanco al 8 para el negro (de 0000 a 1000 en código binario), con números intermedios que representen las tonalidades comprendidas entre ambos extremos. Así, en el ordenador, el dibujo se convierte en una sucesión de números, y todos los datos visuales son introducidos en la memoria, que puede contener millones de imágenes. La ventaja de este sistema es que, una vez dibujada una figura, ya no hay que volver a hacerlo. El ordenador está programado de forma que puede tomar de la memoria y proyectar en una pantalla una imagen o sucesión de imágenes cualesquiera; así, las distintas figuras pueden utilizarse de nuevo, o de forma distinta.

El Flair *es un ejemplo de calculador que permite al dibujante trabajar de la forma que le es familiar. El Flair no tiene teclado, y para crear una imagen basta con dibujar sobre la superficie de una mesa dotada de sensibilidad electrónica a un punzón especial. Mientras que sobre la mesa no queda señal alguna, las líneas aparecen como por arte de magia en la pantalla que hay frente al que dibuja.*

El mismo punzón puede suplir toda una serie de pinceles y realizar trazos tan finos como una línea televisiva o tan grandes como toda la pantalla. Para cambiar de «pincel», o para tener, por ejemplo, una «plumilla» adecuada a la letra cursiva, basta únicamente con reprogramar el calculador pulsando las teclas de selección de pincel que están dispuestas en el borde de la mesa electrónica.

La mesa está hecha de forma que comunica al calculador el punto sobre el cual el punzón utilizado por el dibujante se está moviendo. El utensilio envía una onda electromagnética horizontal y verticalmente a través de la superficie de la mesa electrónica. La posición del punto de contacto es calculada por el calculador midiendo los tiempos de recorrido de las dos ondas desde su punto de partida.

Para los difuminados de color hay un «aeropin-

cel» electrónico. Una vez seleccionado, el aeropincel empieza a «rociar» el color electrónico sobre la pantalla en cuanto el punzón toca la mesa. Cuanto más tiempo permanece el aeropincel en un punto, más intenso se vuelve el color. Los aeropinceles se usan normalmente para crear, junto con los difuminados, el efecto de una superficie tridimensional. En esencia, el aeropincel es un generador de números aleatorios, y su acción no consiste en otra cosa que producir una nube de puntos de color alrededor de un punto determinado.

La utilización de todas estas posibilidades ofrecidas por la pluma y el pincel electrónicos exige del dibujante la misma capacidad que cuando usa pinceles, papel y colores normales, aunque algunos aspectos de su tarea se vean facilitados. Por ejemplo, puede unir dos puntos con una línea recta simplemente marcando los extremos de la línea y dejando que el Flair realice la unión; análogamente, puede ordenar al Flair que haga aparecer en la pantalla figuras geométricas, como círculos o elipses. Basta con precisar los elementos clave, como el centro y el radio, y el Flair hace lo demás. Para seleccionar el color, el dibujante hace aparecer en la «paleta» un cuadro formado por 256 bloques de color, luego hace coincidir la posición del punzón con la del color elegido y presiona con él para introducir el color en el calculador.

El Flair tiene una memoria computerizada para el almacenamiento de los datos visuales en forma digital.

La imagen del Flair está formada por 576 líneas televisivas, cada una de las cuales está subdividida en 768 puntos de color. El calculador almacena los datos procedentes de cada punto en forma de 256 combinaciones de señales rojas, verdes y azules, que en la pantalla aparecerán como un solo color.

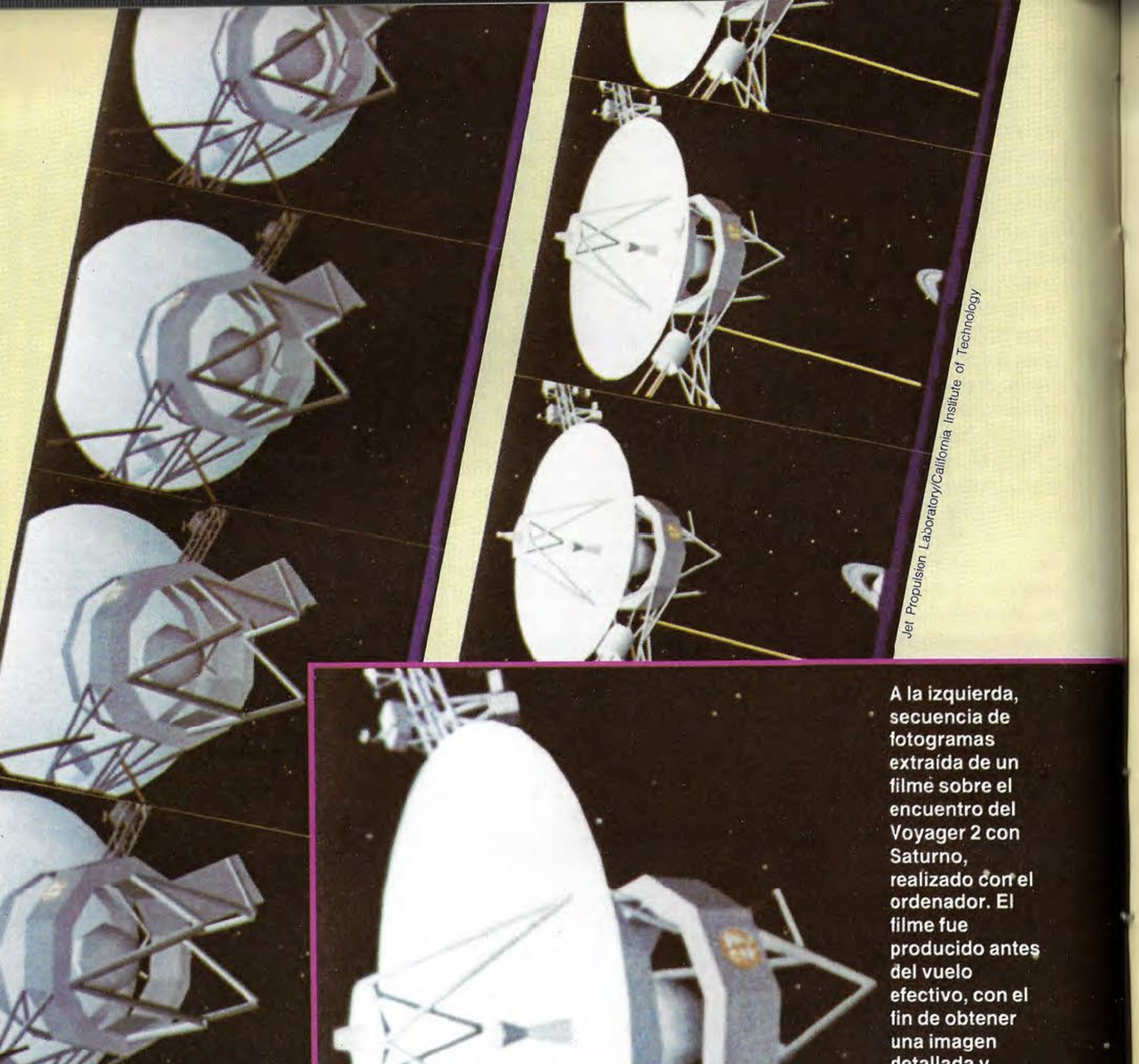
Cuando el dibujante elige un color y mueve el punzón, los movimientos de éste introducen el número adecuado en cada uno de los puntos por los que pasa.

La entrada de los datos y la generación de la imagen son controlados por el único microprocesador 8085A. A él, además de la mesa, hay conectada una unidad de memoria de disco. Así, cuando el trabajo está completo, el dibujo puede ser almacenado en la memoria de disco.

Tablero gráfico conectado con el ordenador personal. El dibujo efectuado con el punzón aparece en la pantalla.



B. Coleman/Marka



Jet Propulsion Laboratory/California Institute of Technology

A la izquierda, secuencia de fotogramas extraída de un filme sobre el encuentro del Voyager 2 con Saturno, realizado con el ordenador. El filme fue producido antes del vuelo efectivo, con el fin de obtener una imagen detallada y convincente de la misión. En el recuadro, el Voyager 2.

Reconociendo que el cine constituye la mejor manera de difundir una gran cantidad de información, en Estados Unidos la NASA y el Jet Propulsion Laboratory utilizan calculadores para producir secuencias visuales animadas de las trayectorias que las sondas propulsadas por cohetes recorrerán en el espacio. Ello requiere toda la potencia de un gran ordenador, pero los

resultados son matemáticamente exactos y pueden mostrar lo que ocurriría si cambiase un parámetro del vuelo. Los resultados son muy nítidos, y los detalles de sombras, perspectivas y estructuras hacen que estas animaciones sean sumamente convincentes.

Para crear imágenes más reales, el personal del ordenador del Jet Propulsion Laboratory recurre a la colaboración de artistas como David Miller, que ha adoptado con entusiasmo el nuevo medio electrónico. Normalmente, Miller trabaja por la noche, porque es entonces cuando puede disponer del calculador para él solo. Sus creaciones son imágenes estáticas computerizadas, figuras insólitas que despiertan admiración, y que son obtenidas con elementos que difícilmente podrían suplirse con las técnicas tradicionales del arte figurativo.

El corazón del sistema ICON, utilizado por la BBC y la ARD (las redes de televisión inglesa y alemana), es un minicalculador usado ya en muchas otras aplicaciones, desde el control de la contabilidad a la transmisión por télex. La programación del calculador se efectúa en el lenguaje denominado RTL2.

Además de aceptar los datos que llegan, el minicalculador genera las señales que producen la imagen televisiva, que está dividida en una serie de puntos o celdillas a lo largo de cada línea televisiva. El ICON funciona con una definición de 1.024 puntos por cada una de las 574 líneas que forman la imagen. Cada punto puede visualizarse en uno cualquiera de los 64 colores que ofrece la «paleta electrónica»: La paleta puede ser accionada dinámicamente por el calculador; de este modo, el color número 6, por ejemplo, una vez puede ser verde mar y, al instante siguiente, azul turquesa. Esto permite al sistema evidenciar partes ocultas de una imagen con un simple cambio de color que haga resaltar una zona concreta en contraste con el color distinto del fondo.

En el ICON, la descripción de una figura no se almacena en forma de números que indican el color de cada punto. Puesto que normalmente los colores son los mismos en zonas amplias, es más fácil representar la imagen introduciendo el color X para un trecho de Y puntos; es lo que se denomina código de «longitud de secuencias», el cual permite una notable reducción de los datos que son necesarios a fin de reproducir una imagen determinada.

Una imagen simple, almacenada con el código de longitud de secuencia, requiere solamente un treintaavo de los datos que serían necesarios si se almacenara un número de color para cada punto. Con el código de longitud de secuencia, el procesador ha de calcular sólo las zonas en que se producen cambios de color. Cuando la imagen es leída a ritmo televisivo, los contadores electróncios digitales examinan los puntos de color y en el momento adecuado accionan el cambio de los números de color. Finalmente, la «paleta» traduce los números de los colores en una serie de tensiones eléctricas correspondientes a los rojos, verdes y azules que constituyen los componentes de cada color concreto en la pantalla.

Cuando se necesita crear animaciones más complejas, como en los largometrajes o en las secuencias espaciales de la NASA, las imágenes se producen de una en una y, una vez completadas, se impresionan en película o se graban en cintas de video. Los datos digitales son suministrados a la memoria de acceso al azar a ritmo relativamente bajo, para ser luego leídos a las altas velocidades requeridas por la imagen televisiva. Esto significa que el calculador puede constituir la imagen a su ritmo y, cuando cada cuadro está completo, puede fotografiarse en la película. Luego los fotogramas son proyectados a la velocidad de 25 por segundo para dar la sensación de movimiento. Esta técnica puede servir para crear imágenes notablemente complejas, y ha sido profusamente utilizada en películas como La guerra de las galaxias *y* Star Trek.

Tal vez la característica más interesante del calculador electrónico es la posibilidad de construir modelos de personas, vehículos, máquinas o edificios en perspectiva tridimensional. El calculador puede programarse para diseñar modelos cuya superficie pueda ser coloreada o decolorada con dibujos y aparezca lisa o rugosa, así como para realizar croquis de objetos vistos desde cualquier ángulo, de cerca o de lejos. Se puede incluso hacer que los objetos aparezcan como si estuvieran iluminados desde un punto determinado con una fuente de luz de cualquier color e intensidad.

Por medio de ciertas elaboraciones, los modelos producidos por calculador pueden situarse en un paisaje, también creado por procesador, utilizando efectos capaces de imitar cualquier condición atmosférica, como, por ejemplo, la niebla o la nubosidad.

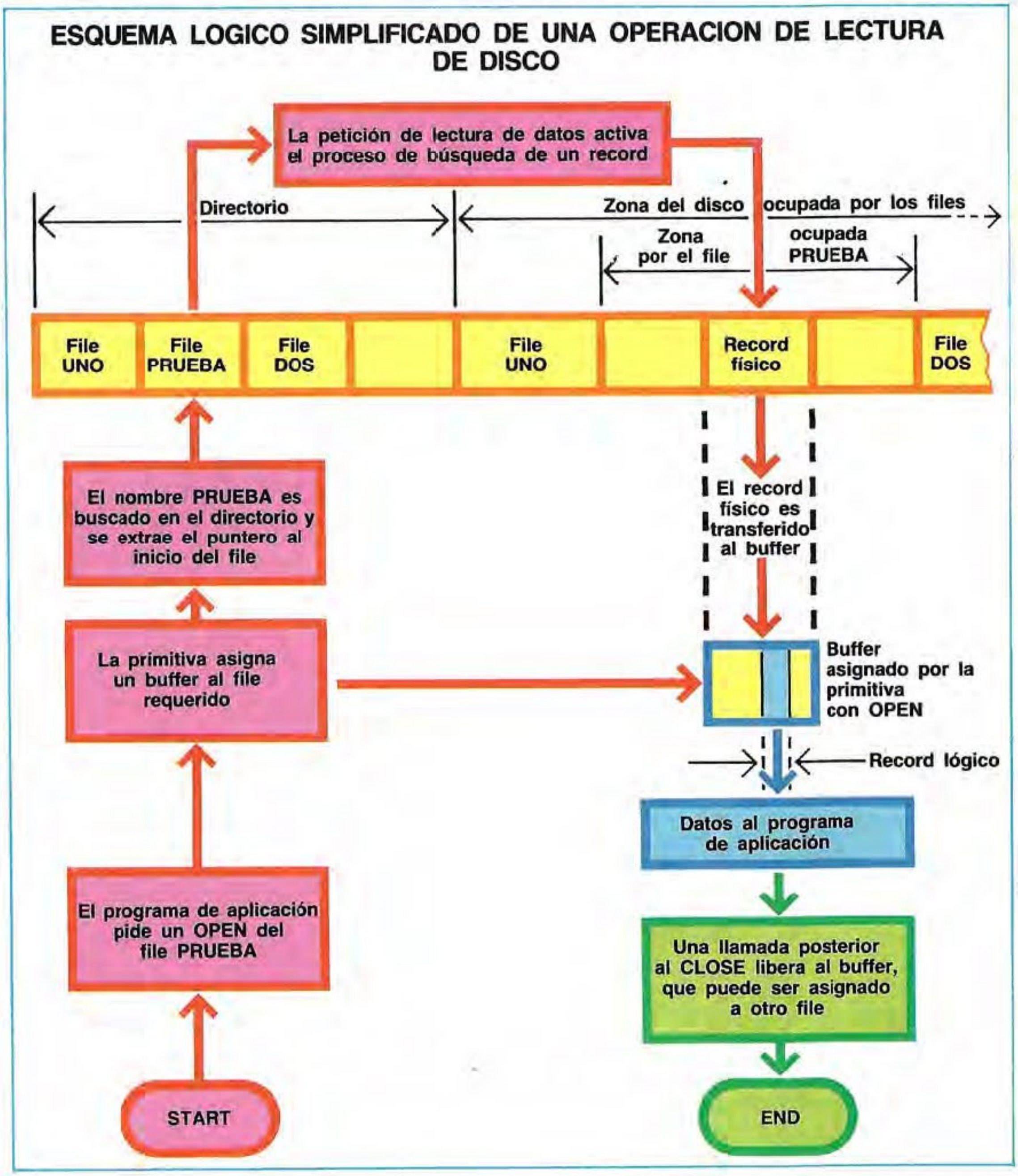
ce el lenguaje Assembler), sino que tiene lugar mediante los intérpretes y compiladores, o con las funciones de comando del sistema operativo. El esquema lógico de las dos formas de acceso se muestra en la pág. 299.

Las principales primitivas disponibles en los sistemas operativos son las siguientes:

SEARCH: Búsqueda de un file dados su nombre y apertura. Con esta primitiva el contenido del file se vuelve accesible. En general, antes de las operaciones de lectura o escritura hay que «abrir» el file, es decir, declarar accesible su contenido.
OPEN:

CLOSE: Cierre del file. Tras haber pedido esta primitiva, el file ya no es accesible. Para elaborar nuevamente sus records, hay que volver a abrirlo.

RENAME: Cambia el nombre de un file.



READ: Lee un record.
WRITE: Escribe un record.
SELECT: Selecciona una de las dos unidades de disco (la unidad de disco se denomina también «disk-drive»).

En los programas de aplicación, mediante adecuadas instrucciones en lenguaje simbólico (por ejemplo, Basic), se activan estas primitivas, que se encargan de la gestión del hardware y del intercambio de datos con los propios programas de aplicación.
Especialmente importantes son las primitivas OPEN y CLOSE (los nombres coinciden exactamente con los de dos instrucciones Basic).
Como ya se ha dicho, durante las operaciones de lectura y escritura, la transferencia de datos entre el disco y la memoria tiene lugar por sectores, aunque el record lógico sea más pequeño. Por lo tanto, hace falta una zona de memoria (buffer) que sirva de soporte temporal para los datos en el momento del intercambio con el disco; luego se extrae el record lógico del contenido del buffer (record físico). La primitiva OPEN tiene como objetivo principal el de asignar un buffer a un file. El número de buffers que se pueden usar simultáneamente es limitado (normalmente, 3 o 4); por lo tanto, si se utilizan más de 3 o 4 files, hay que dejar libres algunos buffers con la primitiva CLOSE para poder abrir otros files. En el gráfico de la página contigua se esquematiza el proceso de lectura de un record. La operación de escritura es análoga. Las primitivas, además de ser llamadas por las instrucciones correspondientes en los lenguajes de alto nivel, se utilizan en numerosos comandos del sistema operativo. Estos comandos realizan las principales funciones necesarias para el uso correcto y la gestión del sistema, y se dividen en dos clases: residentes y transeúntes. Recuérdese que el uso de estos comandos sólo es posible mientras se trabaja bajo sistema operativo; generalmente, si se pasa a Basic los comandos no pueden utilizarse de la misma forma. Existen, sin embargo, otras instrucciones que realizan las mismas funciones operando en Basic (o en cualquier otro lenguaje de alto nivel).
En la pág. 306 se muestra un esquema que ilustra las posibles maneras de operar y las respectivas funciones realizadas. Veamos ahora las funciones de las principales instrucciones y algunos ejemplos de su uso. La forma y la simbología con que hay que escribir las instrucciones dependen del sistema operativo concreto que se esté usando, mientras que las funciones realizadas son análogas para todos los sistemas. A continuación se dan, para cada instrucción, las siglas de identificación más usuales.

1 / Cancelación de un file (comando = **ERA**)
Borra un file del disco. Los parámetros a suministrar son los siguientes:
— en qué disco se halla el file (A/B, o bien 0/1, según el sistema;
— nombre del file;
— tipo de file (Basic, de datos, compilado, etc.).
Por ejemplo, si se desea borrar el file de nombre PRUEBA que reside en el disco A y contiene un programa en Basic, la instrucción, en la simbología del sistema operativo CP/M, es:

ERA PRUEBA.BAS

El código ERA (apócope del verbo inglés *erase*) es la forma simbólica con que se activa el comando de cancelación de un file.
El comando, en la forma presentada, borra los files que residen en el disco seleccionado en el momento de la introducción; si el file se encuentra en la segunda unidad (por ejemplo, la B), hay que especificarlo en el comando, que se convierte en:

ERA B:PRUEBA.BAS

donde el símbolo B: indica que el file se halla en el disco B.
En el sistema operativo PCOS, la instrucción equivalente es:

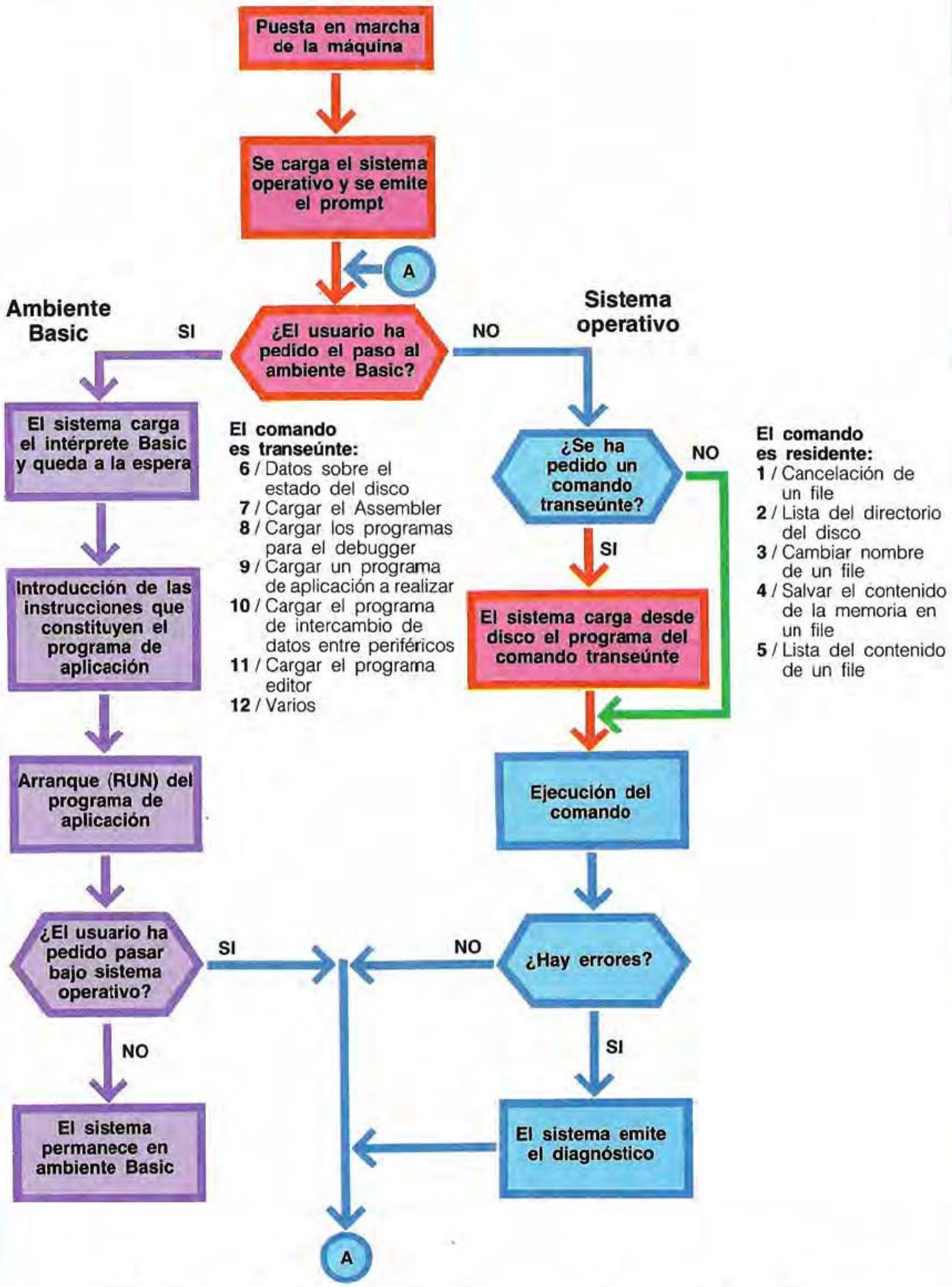
KILL "1:PRUEBA"
(no se requiere la especificación del tipo)

En este sistema, los discos se llaman 0 y 1 (en vez de A y B).
La forma general es:

Código del comando	Unidad de disco	Nombre y tipo del file
ERA	B:	PRUEBA.BAS
KILL	1:	PRUEBA
etc.		

2 / Lista del directorio de un disco (comando = **DIR**)
La instrucción sirve para listar, normalmente en pantalla, el contenido de uno de los dos discos.

ESQUEMA LOGICO DE LAS FUNCIONES DE UN SISTEMA OPERATIVO



Esta división entre comandos transeúntes y residentes es válida para el CP/M

La salida presenta los nombres de los files residentes en el disco seleccionado y su tipo. En algunos sistemas se consigna también la longitud (en sectores o en bytes).
La forma típica de la instrucción es:

DIR.A (para listar el directorio del disco A)
DIR.B (para listar el directorio del disco B)

3 / Cambio de nombre de un file
(comando = **REN**)
Sustituye por otro el nombre de un file. El formato es:

REN B:NUEVO.BAS = PRUEBA.BAS

El nombre PRUEBA del file residente en el disco B se cambia por NUEVO. La forma general es:

Código del comando	Unidad de disco	Nuevo nombre y tipo		Antiguo nombre y tipo
REN	B:	NUEVO.BAS	=	PRUEBA.BAS

4 / Memorización de un programa en disco
(comando = **SAVE**)
El comando SAVE se utiliza en Basic para memorizar en disco (de forma permanente) un programa que reside en la memoria. Las distintas formas de este comando se verán más adelante. La estructura del comando es:

SAVE "A:PRUEBA"

El comando así formado memoriza el programa residente en la memoria en un file de nombre PRUEBA en el disco A.

5 / Lista del contenido de un file
(comando = **TYPE**)
Consigna en pantalla la lista de los files memorizados en formato ASCII. La forma general es:

Código del comando	Unidad disco	Nombre y tipo del file
TYPE	B:	PRUEBA.BAS

La instrucción TYPE B:PRUEBA.BAS efectúa la lista del file PRUEBA.

6 / Estado del disco (comando = **STAT**)
Presenta datos de tipo estadístico correspondientes a la unidad de disco seleccionada, tales como ocupación de los files, espacio aún disponible, etc. Normalmente, el comando no precisa más parámetros que el indicador del disco. La forma general es:

Código del comando	Unidad de disco
STAT	B:

En algunos sistemas (por ejemplo, CM/P), la instrucción STAT suministra también datos sobre los periféricos.

7 / Carga del Assembler (comando = **ASM**)
En los sistemas en los que existe la posibilidad de programar en Assembler, este comando transfiere de disco a memoria un programa Assembler y lo compila traduciéndolo a código máquina (formato hexadecimal, HEX).
La sintaxis es:

Código del comando	Unidad de disco	Nombre del programa
ASM	B:	TEST

El comando ASM B:TEST carga del disco B el programa TEST (que ha de estar escrito en Assembler). La salida del compilador se memoriza en disco con el nombre y tipo TEST.HEX, es decir, con el mismo nombre del fuente pero con tipo HEX (código máquina).

8 / Debugger (comando = **DDT**)
Es una función bastante compleja que permite la comprobación y eventual corrección de los programas de aplicación mientras trabajan.
El debugger permite insertar instrucciones, borrarlas, modificar los contenidos de las memorias, etc. Puesto que la instrucción opera sobre los programas que se están desarrollando, todas las instrucciones han de darse en formato máquina (hexadecimal; un programa cualquiera, para poder realizarse, antes ha de ser traducido del lenguaje de alto nivel a códigos máquina) y todas las respuestas son en hexadecimal. Para la utilización de esta instrucción se exige una notable pericia.

9 / Carga de un programa efectuable
(comando = **LOAD**)
Para tener un programa listo para mandar a ejecución, hay que interpretar o compilar el fuente (escrito, por ejemplo, en Basic). El producto de

LISTA DEL DIRECTORIO DE UN DISCO Y TRANSFERENCIA DE UN FILE

El nombre y el tipo de cada uno de los files contenidos en un disco están memorizados en el directoric.



El comando DIR activa la presentación del directorio del disco en la pantalla. En este caso, el disco seleccionado es el que está inserto en la unidad A.

```
***  BUFFETTI DATA  CP/M.F   vers. 2.02      rev. 820511  ***
A>DIR
A: F1000     COM : GENSIS   COM : ZSID     COM : COB2     FOR
A: STAT      COM : SUBMIT   COM : AZZMAP   COM : AZFILE   COM
A: FORMAT    COM : COPIAB   COM : MAPPAS   COM : CORTUT   COM
A: LEGMAP    COM : NOTE     BAK : OBSLIB   REL : ED       COM
A: XSUB      COM : NOTESYS  BAK : NOTESYS  SUB : NOTE     DOC
A: OBASIC    COM : PIP      COM : BRUN     COM : DDT      COM
A: BASLIB    REL : M80      COM : F80      COM : FORLIB   REL
A: CREF80    COM : LIB      COM : DISKB    COM : CONTRARI BAS
A: WSADMIT   COM : WSMSGSIT OVR : WSOVL1IT OVR : WS       COM
A: DATASTAR COM : WORDSTAR DOC : MERGPRIN OVR : COPIA    COM
A: PLINK-II COM : CODSTAMP COM : BASCOM   COM : BCLOAD
A: MBASIC    COM : L80      COM : CONTRARI REL : CONTRARI COM
A: CONTO     BAS : ORE      BAS : BASLIB   $$$ : REGR     BAS
A>PIP
*B:=REGR

NO FILE: =REGR
*B:=REGR.COM

NO FILE: =REGR.COM
*B:=REGR.BAS
*_
```

En cada fila, al lado del nombre de la unidad de disco seleccionada, aparecen el nombre y el tipo de cuatro files.

Es:a fila informa que en el disco A existen los files F1000 de tipo compilado), GENSIS (compilado), ZSID (compilado), COB2 (fuente escrito en lenguaje Fortran).

En esta fila no se especifica el tipo del file BCLOAD, ya que se trata de un file de datos.

Los files CONTO, ORE, REGR contienen programas fuente escritos en Basic.

En la pantalla también se indica la utilización del comando PIP, mediante el cual se puede transferir un file de una unidad a otra (unidad de disco o periférico).

A la introducción del comando PIP, el sistema responde con el símbolo *, y el operador debe introducir los parámetros complementarios del comando.

El operador pide transferir el file REGR al disco contenido en la unidad B. Como no se especifica el tipo, el sistema no lo encuentra (el file REGR está clasificado como BAS, ver directorio), y emite un diagnóstico.

El operador pide transferir el file REGR.COM. Una vez más, el sistema emite un diagnóstico porque el file solicitado no existe.

La petición del operador es correcta, y el comando se ejecuta.

USO DEL COMANDO STAT

El comando STAT activa la visualización del estado de un file.



```
A: DATASTAR COM : WORDSTAR DOC : MERGPRIN OVR : COPIA    COM
A: PLINK-II COM : CODSTAMP COM : BASCOM   COM : BCLOAD
A: MBASIC   COM : L80      COM : CONTRARI REL : CONTRARI COM
A: CONTO    BAS : ORE      BAS : BASLIB   $$$ : REGR     BAS
A>STAT CONTO

File Not Found
A>STAT CONTO.BAS

 Recs  Bytes  Ext Acc
   5     4k       R/W A:CONTO.BAS
Bytes Remaining On A: 476k

A>STAT
A: R/W, Space: 476k


A>STAT BCLOAD

 Recs  Bytes  Ext Acc
   1     4k     1 R/W A:BCLOAD
Bytes Remaining On A: 476k

A>
```

Después de la presentación del directorio se pide el estado del file CONTO. El sistema lo interpreta (por defecto) como file de datos (no se ha especificado el tipo) y no lo encuentra.

El comando se reintroduce en la forma correcta, y el sistema presenta el estado: el file CONTO contiene 5 records (lógicos), para un total de 4.000 bytes, tiene una sola extensión y permite bien la lectura, bien la escritura (R y W son las iniciales de read y write). Al final se presenta el espacio total todavía disponible en el disco.

Esta última información también puede obtenerse introduciendo un solo comando STAT, sin especificar ningún nombre de file.

Se pide el estado del file de datos BCLOAD. En este caso no es necesaria la especificación del tipo (ver el directorio).

la compilación es un file en disco que contiene los códigos hexadecimales correspondientes a las instrucciones del fuente.
Para trabajar con este programa hay que tomar los códigos del disco y transferirlos a la memoria. Este módulo ejecuta dicha función. La sintaxis es:

Código del comando	Unidad de disco	Nombre del programa
LOAD	B:	PRUEBA

El comando LOAD B:PRUEBA toma el file PRUEBA en versión compilada, es decir, tipo HEX (si del nombre PRUEBA sólo hay el tipo BAS, es decir, fuente, el sistema indica error), y lo transfiere al área de memoria transeúnte, listo para la ejecución.
Hay que señalar que en los sistemas sin compilador la instrucción LOAD efectúa la carga de programas escritos en Basic.
Para cargar un fuente Basic en los sistemas provistos de compilador, antes hay que pasar a Basic y luego utilizar el comando LOAD. En este caso el comportamiento es idéntico al anterior, pero se carga la versión fuente (Basic) del programa.
Resumiendo, el comando LOAD tiene los significados siguientes:

- en los sistemas sin compilador, carga el fuente
- en los sistemas con compilador, carga el programa HEX si se halla bajo el sistema operativo, y carga el fuente en ambiente Basic

10 / Intercambio de datos entre periféricos (comando = **PIP**)
Permite transferir datos o programas de un periférico a otro. Se utiliza sobre todo para el intercambio de programas entre las dos unidades de disco. La sintaxis es:

```
> PIP
* B: = PRUEBA.BAS
```

El código PIP activa el programa de intercambio y el sistema responde con el símbolo *. Entonces se puede introducir la segunda parte del comando, que consiste en especificar entre qué periféricos ha de efectuarse el intercambio.
En el ejemplo, el programa PRUEBA (escrito en Basic y residente en el disco A) se transfiere al disco B (con el mismo nombre).
En este comando se pueden usar todas las formas de nombres y tipos previstas en el sistema operativo. Por ejemplo:

```
> PIP
* B: = *.*
```

Transfiere al disco B todo el contenido de A (los símbolos * en lugar del nombre y del tipo significan «cualquiera»).

```
> PIP
* B: = *.BAS
```

Se especifica sólo el tipo (BAS), por lo que se transfieren todos los files en Basic (cualquiera que sea su nombre).

```
> PIP
* B: = *.
```

El tipo se deja en blanco, lo cual, para el sistema, significa un file de datos: la instrucción efectúa una copia de todos los files de datos del disco A en el disco B.

11 / Editor (comando = **ED**)
Esta función permite leer, cargar en la memoria y luego variar el contenido de un file. Se trata de un programa complejo que realiza numerosas operaciones de corrección (editing), como:
— sustitución, cancelación, inserción de caracteres o de líneas enteras
— búsqueda en un texto (o en un programa) de palabras concretas
— sustitución de grupos de caracteres
Al final de la tarea se crea un nuevo file con los datos correctos.

12/ Varias
Además de las enumeradas, hay otras muchas funciones específicas de cada sistema operativo. Las fundamentales se refieren al formateado de los diskettes –para la cual, aparte del código de llamada (por ejemplo, FORMAT), no hacen falta otros parámetros–, y la copia del sistema operativo en un diskette de reserva.
A título de ejemplo, enumeramos algunas de las principales funciones previstas en el sistema PCOS (M20 Olivetti), sin hacer distinción entre programas residentes y transeúntes.

TEST 8

1 / Enumerar los principales componentes de un sistema operativo para ordenador personal.

2 / ¿Cuál de estas afirmaciones es cierta?
a: la memoria de la máquina está por completo a disposición del usuario;
b: los programas del usuario se cargan en el área de memoria residente;
c: el sistema operativo se carga en parte en el área de memoria transeúnte y en parte en el área residente.

3 / ¿Qué son las tablas de extensión (tablas de los extent) de los files?
a: tablas en las que se consigna la longitud de los files;
b: listas de los files presentes en el disco;
c: tablas con las direcciones de los distintos trozos en que está dividido un file.

4 / ¿Para qué sirve el directorio?
a: para memorizar todos los datos sobre los files;
b: para contener las diversas partes del sistema operativo;
c: para contener los programas de aplicación.

5 / ¿En qué consiste el formateado de discos?
a: en la cancelación de todo su contenido;
b: en su división en pistas y sectores;
c: en controlar si se hallan en buen estado.

Las soluciones, en la pág. 313.

- Formateado de diskettes
- Copia del sistema operativo
- Asignación de una palabra de orden (impide el acceso a los datos por parte de personas no autorizadas)
- Creación de un file
- Asignación de una palabra de orden (password) a un file concreto (de este modo se impide el acceso sólo a ciertos files y no a todo el disco)
- Instrucciones especiales para variar las modalidades operativas de los periféricos (por ejemplo, los formatos de impresión)

Funciones de paso al ambiente Basic

Al conectar la máquina, tras la carga del sistema operativo, el ordenador está listo para funcionar. La máquina está bajo el control del sistema operativo y no puede aceptar instrucciones Basic. Para introducir un programa hay que pasar al ambiente Basic.

La simbología de la instrucción depende del sistema operativo; por ejemplo, en CP/M la instrucción es MBASIC, y en PCOS es BA. Una vez dada esta instrucción de la forma requerida por el sistema operativo concreto, se carga en memoria el intérprete Basic y puede comenzar la tarea de programación. Al final, para volver bajo control del sistema operativo, hay que introducir la instrucción correspondiente, que normalmente es SYSTEM.

Algunos sistemas no necesitan estos pasos, puesto que sólo pueden trabajar en Basic.

Para lenguajes distintos del Basic (por ejemplo, Fortran) no existe un «ambiente» especial. Primero hay que escribir las instrucciones como si se tratara de un file de datos, por ejemplo, usando el programa editor (ED). Al final de la escritura se puede llamar el programa compilador, obteniendo la traducción en código máquina. La forma de programa así obtenida está lista para su ejecución.

El mismo procedimiento se tiene que seguir para la compilación de un fuente en Basic (si el sistema operativo incluye compilador), con la

USO DEL COMANDO FORMAT EN CP/M

El comando FORMAT permite predisponer un disco nuevo para su posterior utilización. La obediencia al comando de un disco ya formateado y grabado determina la pérdida de su contenido. La pantalla muestra qué aparece introduciendo el comando FORMAT con el sistema operativo CP/M. El procedimiento de formateado es guiado atentamente por el SO para minimizar la posibilidad de cometer errores.



```
** FORMAT **   Formatter alta/baja densidad      rev. 03
Buffetti Data Roma 17 mayo 1982

Alta densidad doble cara                1
Baja densidad doble cara                2
BD801 system formatter                  3
Baja densidad simple cara               4

Seleccionar el tipo con el correspondiente código;
para salir pulsar "return";

Unidad seleccionada:                            A
¿Qué unidad contiene el floppy a formatear?     B

¿Desea verificar el floppy (S o N)?             S

Formateado: superficie 01, pista 4C

Verificación: superficie 00, pista 03



Selección: 3
```

El sistema propone al operador cuatro posibilidades, entre las cuales se seleccionará la deseada.

El sistema informa que está operando en la unidad A, y pide al operador en qué unidad está montado el diskette a formatear. La respuesta del operador es B.

El sistema pide si se desea la verificación del floppy. En caso de asentimiento por parte del operador (S), el propio SO controlará, una vez completado el formateado, que todos los sectores del floppy estén en buen estado.

Esta fila informa, a medida que se va realizando el formateado, hasta qué punto se ha llegado. Las superficies son dos (00 y 01) y las pistas por superficie 4C (en hexadecimal).

Esta fila aparece una vez completado el formateado (si se ha pedido la verificación del floppy) e informa en qué punto está la operación de verificación. La fotografía se obtuvo mientras el sistema verificaba la pista 3 de la superficie 00. El trabajo terminará en la pista 4C de la superficie 01.

SOLUCIONES DEL TEST 8

1 / Las principales funciones son:

– gestión de las operaciones I/O (discos, impresoras, etc.). Forman parte de este bloque todos los programas que "interpretan" las instrucciones de entrada/ salida (I/O) y activan los controles y demás funciones para gobernar los periféricos. Cada módulo destinado a un periférico se denomina "primitiva".

– intérprete de las instrucciones. Es el módulo que analiza las instrucciones introducidas mediante el teclado y las hace ejecutar.

Estos dos componentes del sistema operativo realizan las funciones básicas y han de estar siempre presentes (área de memoria residente). Hay otras muchas funciones (lista del directorio, intercambio de datos entre periféricos, etc.) que no son de uso continuo. Sus programas respectivos residen en disco y se cargan en el área de memoria transéunte cuando el usuario lo requiere.

2 / c: la explicación va incluida en la respuesta a la pregunta **1.** La memoria de la máquina se divide entre los programas de sistema (residentes) y los programas del usuario. Además, en el caso del Basic interpretado ha de cargarse en memoria también el intérprete. Por el contrario, en el Basic compilado sólo se carga el programa de usuario, puesto que, estando ya traducido (en la fase de compilación) a lenguaje máquina, no necesita intérprete. De ello se deriva otra ventaja de la compilación: puesto que no hay que cargar el intérprete, queda mayor área de memoria a disposición del usuario, ventaja nada despreciable si se tiene en cuenta que el intérprete puede ocupar unos 12.000 bytes (12 kbytes). Así, en una máquina de 64 kbytes de memoria total, la distribución es:

Sistema operativo	unos 16 k	(depende de la complejidad del sistema operativo)
Intérprete	12 k	
Area usuario	36 k	

Para el Basic compilado, al no haber intérprete, el área de usuario aumenta a 48 k.

3 / c: un file, tanto si contiene datos como programas, no conserva su longitud inicial. Las sucesivas actualizaciones de su contenido pueden requerir un número de registros mayor del que tiene el file. En tal caso, los nuevos registros se escriben en zonas del disco apartadas del file al que pertenecen; para "unir" lógicamente los distintos trozos se usa la tabla de extensiones, que consigna, para cada file, la posición física de las diversas extensiones (adiciones).

4 / a: el directorio es una zona del disco, de uso reservado al sistema operativo, en el que se memorizan los datos sobre el contenido del disco.

5 / Las tres respuestas (a, b y c) son válidas consideradas en conjunto. El principal objetivo del formateado es dividir el disco en pistas y sectores, para crear un método de direccionamiento del espacio físico. Durante esta operación, todo el contenido del disco se borra. Además, en muchos sistemas operativos hay prevista una función de comprobación, que consiste en la escritura y relectura de todo el disco. De esta forma se evidencian eventuales zonas defectuosas.

ventaja de que el programa puede ser probado en forma interpretada y los errores pueden corregirse antes de la compilación. Por el contrario, en los lenguajes sin intérprete hay que recompilar el fuente tras cada modificación.

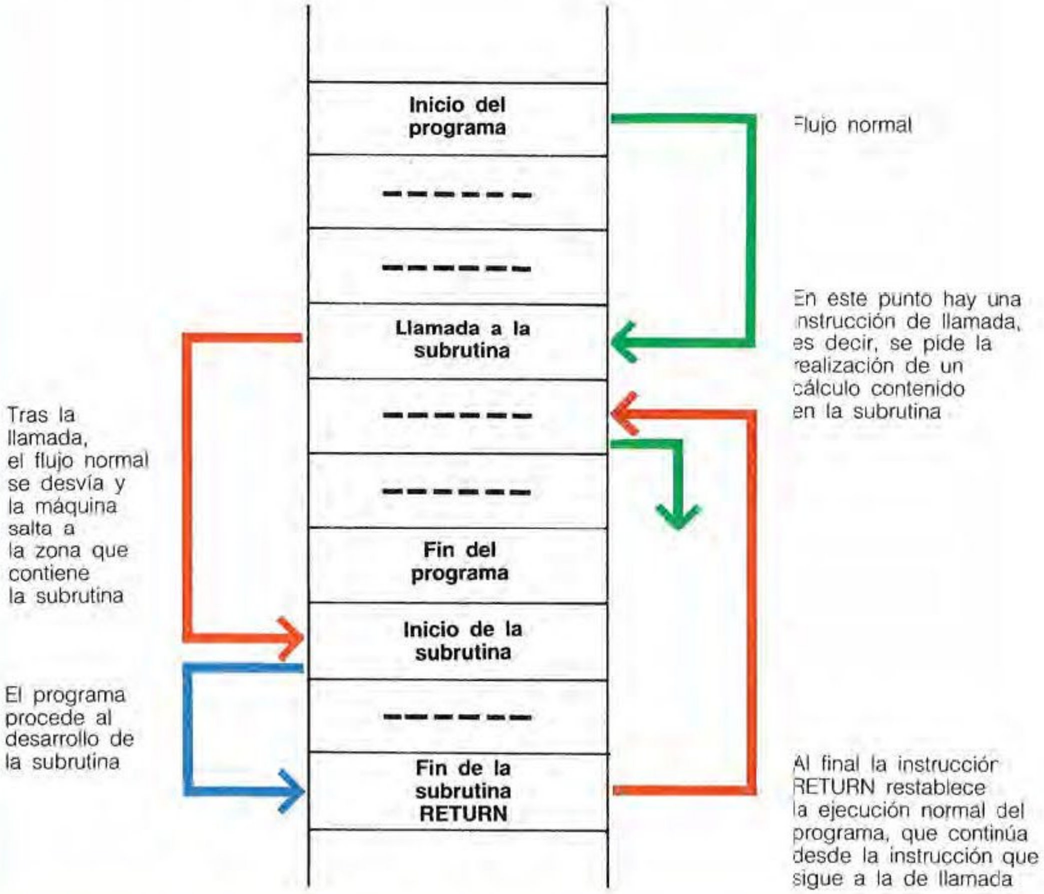
Criterios de evaluación de un sistema operativo

Las principales funciones de un sistema operativo constituyen un «metro» en el que basarse para una evaluación, aunque aproximativa.
Un sistema operativo es, en general, muy complejo, y para evaluarlo hacen falta experiencia y conocimientos en profundidad. En el software pueden esconderse errores nunca evidenciados, por ejemplo, porque nunca se haya dado, en las aplicaciones, una determinada coincidencia de hechos que lleva a un error. Para algunos valores de los datos podría activarse dicha situación, y un programa que parecía funcionar bien resulta defectuoso. La comprobación del software es muy compleja y, aun usando técnicas de control sofisticadas, no es seguro que salgan a la luz todos los defectos. Una evaluación aproximada se puede efectuar en base a las funciones que contiene el sistema operativo. En general, es prioritaria la presencia del compilador Basic; el resto ha de evaluarse caso por caso en función de las necesidades.

Subdivisión de los programas

En la escritura de programas complejos hay que tener en cuenta dos factores determinantes: evitar escribir varias veces las mismas instrucciones y mantener el programa entero dentro de la amplitud máxima del área de usuario (que es parte del área transeúnte; si el programa está en Basic interpretado, una parte considerable del área transeúnte estará ocupada por el intérprete).
Muchos cálculos y funciones iguales pueden requerirse en distintos puntos de un mismo programa, aunque operen con valores numéricos diferentes. Reescribir estos cálculos cada vez que son necesarios constituye un despilfarro inútil de memoria; conviene, por el contrario, preparar una versión única de cada procedimiento de cálculo escribiéndolo en forma generalizada, para luego poder utilizarlo en todos los cálculos similares presentes en el programa. Una parte de programa estructurada de esta forma se llama **subrutina** o **subprograma**. Para obtener el desarrollo de los cálculos contenidos en una subrutina, basta con «llamarla» en el curso del programa mediante las oportunas instrucciones en lenguaje simbólico.
A nivel de máquina, la llamada provoca un salto desde el punto del programa en que se desea utilizar la subrutina (punto de llamada) hasta el punto donde comienza la subrutina requerida. Al final de la subrutina ha de hallarse la instrucción RETURN (retorno); al reconocerla, el control vuelve a la instrucción que sigue a la de llamada, y prosigue normalmente hasta el final del programa. Desde el punto de vista lógico, es como si tuviéramos un «bloque» aparte (subrutina) que se «inserta» una vez tras otra en los puntos en que hace falta. En el gráfico de la página contigua se muestra el esquema lógico de la llamada a una subrutina.
En los diagramas de flujo, la llamada a una subrutina se indica, normalmente, con el símbolo de acción genérica (rectángulo) especificando qué subrutina se llama. En los sistemas operativos más complejos, las subrutinas pueden no formar parte del programa. Residen normalmente en discos y son cargadas en memoria cuando hacen falta. Durante la carga pueden ocupar el lugar de otra subrutina que en ese momento no se utiliza; en este caso se produce una «superposición» (en momentos sucesivos) de diversas subrutinas en la misma área de memoria, con una ocupación muy inferior a la que sería necesaria si todas las subrutinas estuvieran simultáneamente en la memoria. Hay ejemplos de subrutinas en las funciones matemáticas incluidas en los lenguajes de alto nivel. En Basic, como en los demás lenguajes, se pueden realizar cálculos complejos (por ejemplo, raíces cuadradas, logaritmos, etc.) mediante subrutinas de sistema, llamadas más propiamente funciones. El módulo que realiza un cálculo concreto funciona igual que una subrutina: cuando el usuario lo requiere, la máquina «salta» al punto que contiene la subrutina de cálculo, la ejecuta y luego vuelve a emprender el programa aplicativo.
La manera «simbólica» de utilizar las subrutinas de sistema es distinta de la empleada para las subrutinas escritas por el usuario, pero la mecánica es la misma. Normalmente, para llamar una función de sistema basta con indicar su nombre (así, la raíz cuadrada se llama con el símbolo SQR), mientras que para las subrutinas del usuario hace falta una instrucción específica que prepara el proceso de «salto» y retorno.

MECANISMO DE LLAMADA DE UNA SUBRUTINA (CONTENIDA EN EL PROGRAMA)



Un factor determinante a efectos del empleo racional de un sistema de elaboración, es la ocupación de la memoria. El correcto análisis del problema en fase de planteamiento y el uso de las subrutinas pueden reducir el número de instrucciones necesarias para completar el programa; pese a ello, se llega con frecuencia a una ocupación excesiva de la memoria. El inconveniente puede evitarse mediante dos técnicas similares: la «concatenación» y la activación de un programa desde otro programa.

La concatenación o encadenamiento tiene aspectos similares a la gestión de subrutinas no residentes, y consiste en cargar en memoria algunas partes del programa sólo cuando se utilizan, sustituyendo unas por otras. A diferencia de los sistemas más evolucionados, en los microordenadores ha de ser el usuario quien gestione todas las transferencias.

En el gráfico de la pág. 316 se muestra el esquema de ejecución de una concatenación. El programa de usuario está dividido, por ejemplo, en tres partes, denominadas respectivamente PROG-1, PROG-2 y PROG-3. La primera parte (PROG-1) es el «cuerpo» principal y reside permanentemente en la memoria; las otras dos se cargan con la operación de encadenamiento (CHAIN) sólo cuando son necesarias.

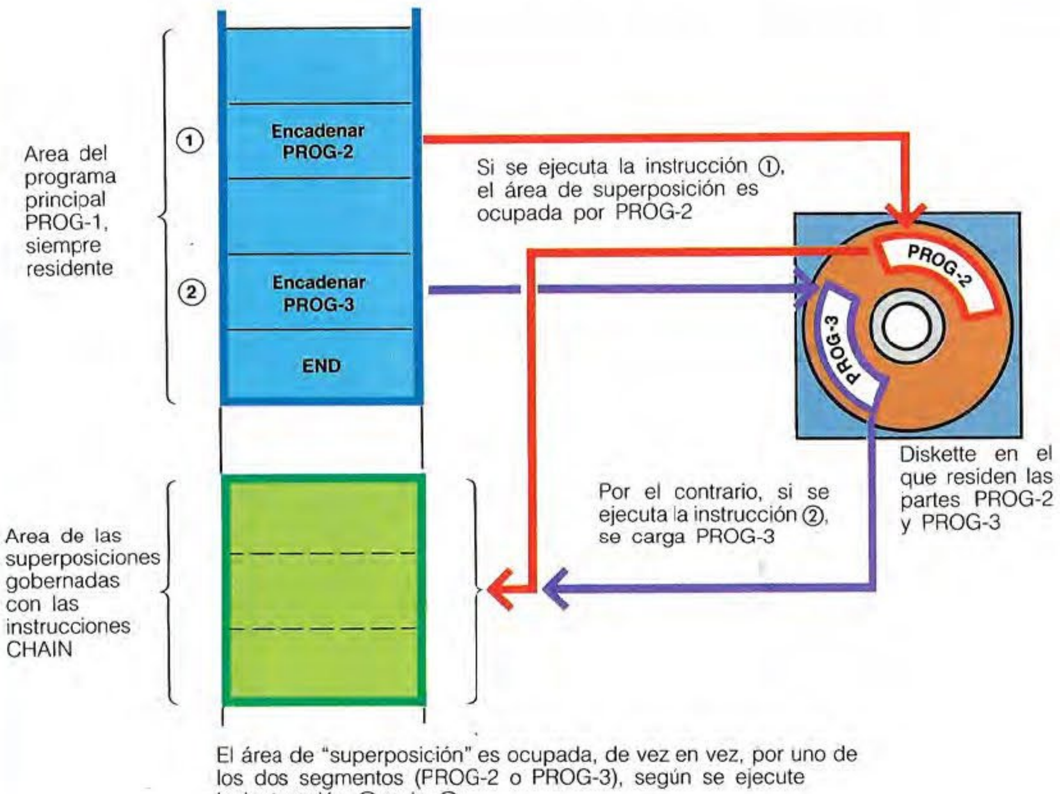
El segundo método consiste en activar la ejecución de un programa mediante una instrucción

(RUN) contenida en el programa en curso. En este caso, la memoria no conserva rastro alguno del programa primitivo: el bloque llamado sustituye por completo al anterior.

Transferencia de parámetros

Para que un programa pueda funcionar, todos los puntos del mismo han de «conocer» las magnitudes utilizadas. Por ejemplo, al escribir un programa para calcular el área de un círculo hay que introducir el valor del radio. Si este programa se recubre con otro módulo, el valor del radio se pierde, a no ser que el programador transfiera dicho valor al nuevo módulo, es decir, opere una transferencia de parámetros. Ésta tiene lugar de forma diferente según la técnica utilizada para escribir las distintas partes del programa (subrutinas, CHAIN, RUN).
En las subrutinas (salvo en casos que se mencionarán cuando se exponga el lenguaje Basic) no hace falta ninguna instrucción especial.
Las subrutinas (en la forma estándar del Basic) son partes integrantes del programa principal, y por tanto «conocen» todos los valores (parámetros) como el programa mismo.
La operación CHAIN requiere una instrucción especial (COMMON) que declare cuáles son los valores de uso común entre los diversos módulos concatenados. Con la instrucción RUN no se puede pasar ningún parámetro sino utilizando el disco como memoria de apoyo temporal. Los valores a transferir son escritos en disco por el programa «llamador» (que contiene la instrucción RUN) y leídos, tras su carga, por el programa llamado. Las instrucciones que hemos visto para la segmentación de programas son típicas del Basic, y en el capítulo dedicado a la sintaxis de dicho lenguaje serán estudiadas con detalle. En otros lenguajes, incluso en el Fortran, los métodos son formalmente distintos. Las principales variantes se expondrán en los capítulos dedicados a los diversos lenguajes.

ESQUEMA DE EJECUCION DE UN ENCADENAMIENTO



El área de "superposición" es ocupada, de vez en vez, por uno de los dos segmentos (PROG-2 o PROG-3), según se ejecute la instrucción ① o la ②

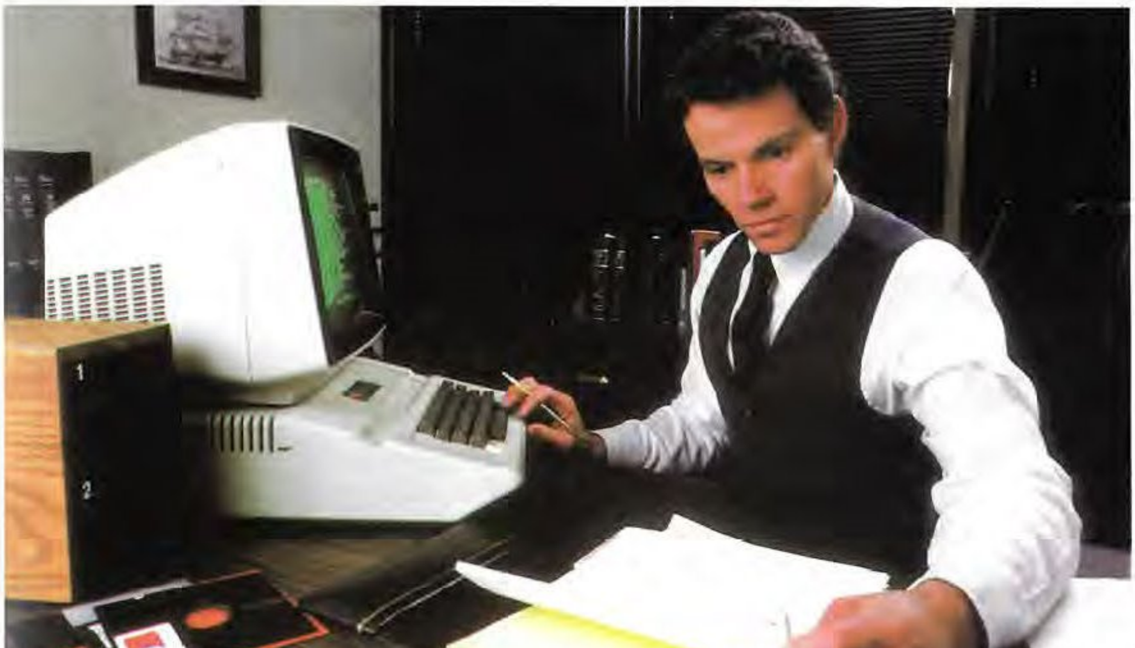
El lenguaje de la programación Basic

Los únicos medios para dialogar con los microordenadores son el teclado, como dispositivo de entrada, y la pantalla (o la impresora) como dispositivo de salida. Todas las instrucciones han de introducirse mediante el teclado, y la eventual respuesta de la máquina aparece en la pantalla o es escrita por la impresora. La introducción mediante teclado se efectúa escribiendo la instrucción como si se tratara de una máquina de escribir normal. La operación de introducción por teclado se denomina «digitación», puesto que cada símbolo se convierte en una señal digital. Al final de la introducción hay que pulsar la tecla CR (en algunos teclados, RETURN o ↵). Supongamos, por ejemplo, que nos hallamos en sistema operativo y deseamos la lista de los ficheros contenidos en el disco. La instrucción a introducir es DIR; por tanto, tendremos que digitar (componer en el teclado) las letras (DIR) y luego pulsar la tecla CR (o RETURN o ↵, según el tipo de teclado). La tecla CR permite informar a la máquina de que hemos terminado de componer la instrucción y puede proceder a su ejecución.

Todo lo que introducimos mediante el teclado aparece en la pantalla: así, mientras escribimos la instrucción DIR, en la pantalla van apareciendo las letras a medida que son tecleadas; de esta forma se puede seguir y comprobar la exactitud de lo que se va escribiendo. Al final de la frase el cursor se detiene junto a la última letra. Al pulsar la tecla CR, el cursor se posiciona al comienzo de la línea siguiente y la instrucción digitada es ejecutada. La secuencia se muestra en el gráfico de la pág. 318.

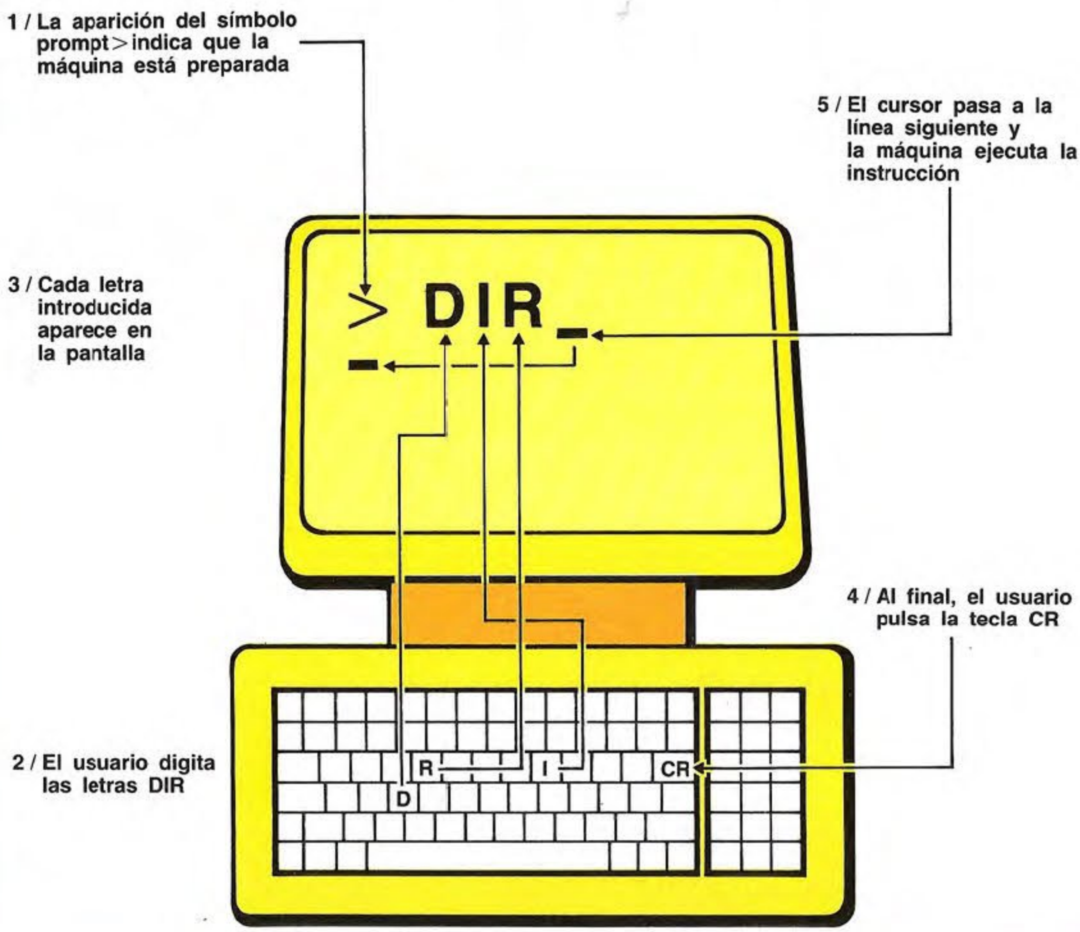
Esta forma de funcionamiento, con escritura en pantalla de los caracteres tecleados, se denomina «con eco» (lo que aparece en la pantalla es el eco de lo que se teclea), y es la forma normal de operación. En algunas circunstancias, por ejemplo durante la introducción de una palabra de instrucción, se puede suprimir el

El ordenador realiza también su valiosa función en los despachos de los ejecutivos.



Marka

INTRODUCCION DE LA INSTRUCCION DIR

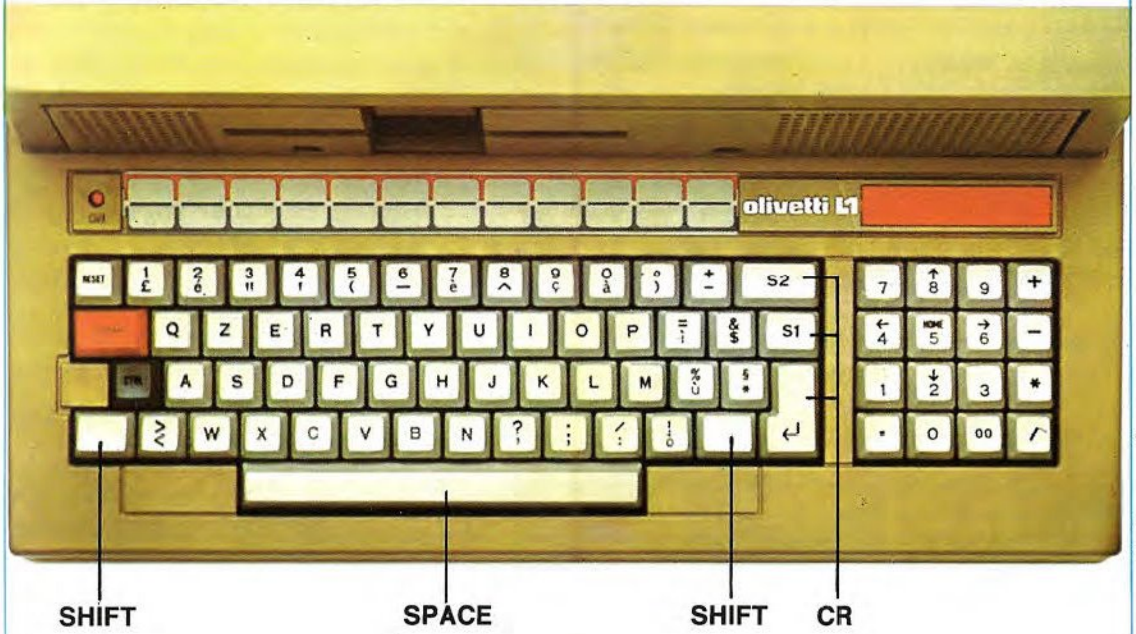


eco. Lo que se escribe en el teclado no aparece en la pantalla, y un eventual observador no podría leer lo que estamos tecleando. En la pág. 319 se muestran dos formas típicas de teclado. La disposición de las letras es la misma en todos, mientras que algunas teclas especiales pueden faltar o tener una disposición diferente. El primer teclado, por ejemplo, tiene 15 teclas marcadas con las siglas F1 a F15 (teclas de función), que pueden ser programadas para realizar funciones concretas. Si se desea, se puede asociar una de estas teclas a una palabra de instrucción, que aparecerá en la pantalla al pulsar la tecla, evitando la necesidad de teclear cada vez todas las letras. En algunos teclados puede haber teclas no utilizadas, previstas para otro tipo de ordenador o para aplicaciones distintas (ver las teclas en rojo en las fotografías de la página contigua).

En todos los teclados hay una barra de espaciado (SPACE), que realiza la misma función que en una máquina de escribir, es decir, la intercalación de espacios en blanco (recuérdese que, para un ordenador, un espacio en blanco equivale a un carácter y, como tal, ocupa un espacio en la memoria).

Las letras aparecen en las teclas en mayúscula, puesto que las instrucciones se componen normalmente de letras mayúsculas, pero los teclados incluyen también las minúsculas; para escribir en minúsculas basta con mantener apretada la tecla SHIFT mientras se teclean las letras. Esta forma de operar puede resultar incómoda si hay que escribir muchas minúsculas segui-

TECLADO DEL ORDENADOR OLIVETTI M20



Olivetti

TECLADO DEL ORDENADOR BUFFETTI B2/801



Dominique Fradin

das, puesto que ocupa ambas manos; en tal caso se puede bloquear la selección de minúsculas pulsando la tecla LOCK (para volver a las mayúsculas basta con pulsar de nuevo LOCK). Algunas teclas tienen una doble simbología: el símbolo inferior es el normal, mientras que para digitar el símbolo superior hay que pulsar a la vez la tecla SHIFT. La SHIFT (selección momentánea de minúsculas) tiene también la función de seleccionar los caracteres de las teclas dobles. Hay, por último, dos teclas especiales, ESC y CTRL, cuya activación provoca el envío de caracteres especiales con funciones de control (las funciones y aplicaciones de estas teclas se verán más adelante).
Resumiendo, las teclas presentes en el teclado de un microordenador son:

- **alfabéticas y numéricas**: para la introducción de datos;
- **funcionales**: su función ha de determinarse mediante programa;
- **especiales** (ESC, CTRL): realizan funciones especiales;
- **no usadas**: presentes para otras aplicaciones del teclado;
- **tecla CR**: para terminar un dato y llevar a comienzo de línea el cursor;
- **tecla SHIFT**: minúsculas o caracteres superiores de las teclas con dos símbolos;
- **tecla LOCK**: bloquea o libera el estado SHIFT (minúsculas).

La activación de una tecla cualquiera provoca la generación del correspondiente código ASCII. Al final de la instrucción, con la introducción del código CR (tecla CR), los caracteres (codificados en ASCII) son enviados a la unidad central y «procesados» (o sea, analizados) según el programa en curso en ese momento. Algunos símbolos (como la barra) son válidos en Basic, mientras que no son reconocidos por algunos sistemas operativos. En tal caso, la máquina emite un diagnóstico, evidenciando el error y pidiendo una nueva introducción.

Generalidades

El lenguaje Basic (Beginner All-purpose Symbolic Instruction Code) surgió en 1963, desarrollado por Kemmeny y Kurts, en forma interpretada y con una dotación de instrucciones muy restringida. Estaba destinado a usos concretos, por parte de usuarios no expertos en programación, y tenía muchas limitaciones. En pocos años, sin embargo, ha conocido un enorme desarrollo, en parte debido a la cada vez más amplia difusión de los microordenadores, hasta alcanzar el nivel de un lenguaje completo, compilado, válido como alternativa a los lenguajes preexistentes (Fortran, Cobol, RPG, etc.). En su evolución ha conservado sus características originarias de sencillez y comprensión inmediata; es, por tanto, el lenguaje más adecuado para introducirse en el mundo de los ordenadores. La especial estructura y la gran variedad de instrucciones del Basic moderno hacen de él un lenguaje útil (y muy utilizado) también para los profesionales de la informática. El estudio del Basic además suministra los medios para desarrollar cualquier aplicación, científica o administrativa, y no sólo con microordenadores, ya que casi todas las grandes máquinas pueden operar en Basic. Como todos los lenguajes, el Basic ha de responder a normas que describen sus funciones y modalidades operativas.
Los productores de software y de hardware han de respetar las especificaciones expresadas en las normas, y complementar la forma base con eventuales funciones especiales.
La versión del Basic que veremos seguidamente es la estándar (definida en el documento oficial ANSI-BSRX 3.60 - 1978, redactado por el organismo estadounidense para la definición de estándares) con los complementos del moderno Basic 80. Al final se describirán algunas formas especiales adoptadas por los fabricantes más conocidos, así como las características del Basic compilado.

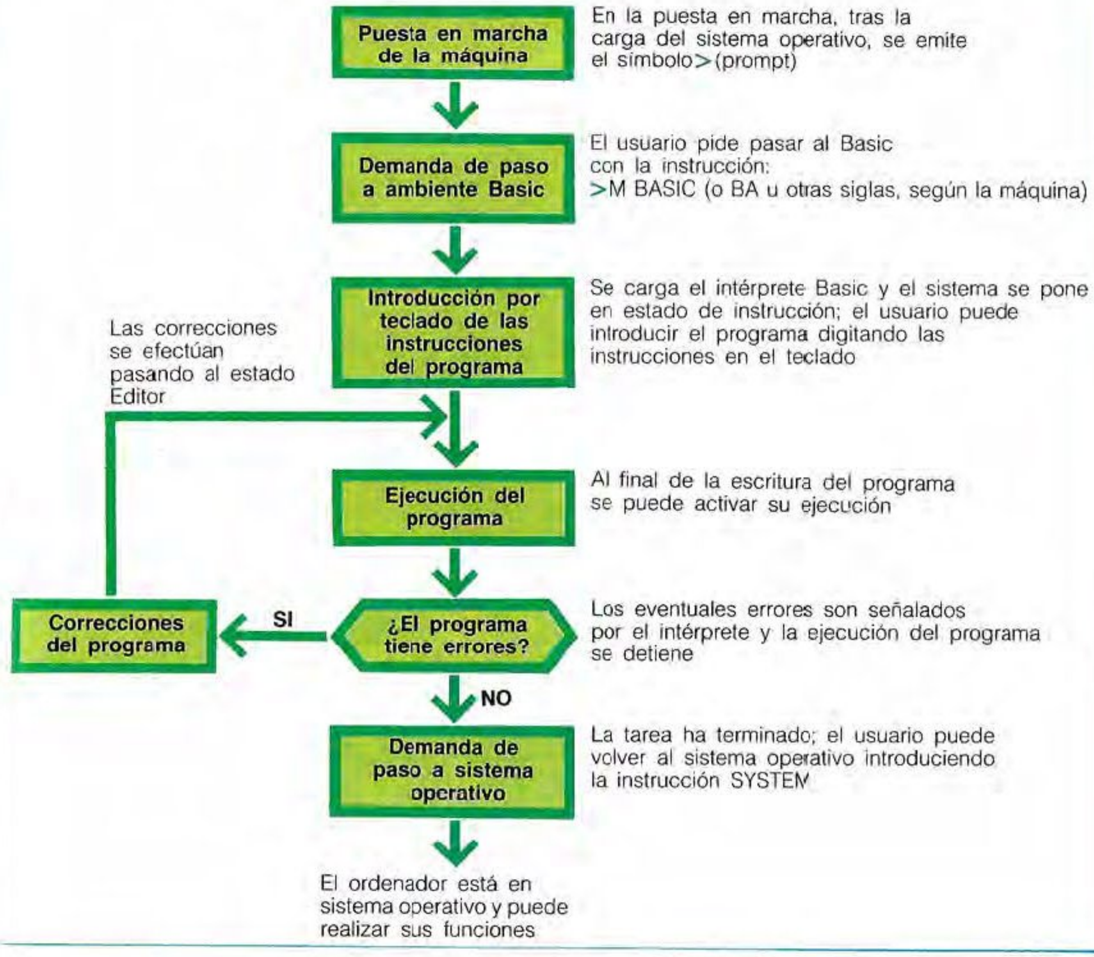
Modalidades operativas

El Basic interpretado tiene tres modalidades operativas (ver teclados en pág. 319): **estado de instrucción, estado de ejecución** y **estado de editor**. El estado de instrucción permite introducir todas las funciones de instrucción, como la petición del listado de un programa, las funciones que operan en los ficheros, la introducción de un programa, etc. En el estado de ejecución se pueden activar los programas de aplicación o efectuar cálculos inmediatos. El estado de editor sirve para efectuar correcciones en los programas de aplicación. En los gráficos de la página contigua se enumeran los pasos a efectuar a partir de la puesta en marcha de la máquina para la escritura y para la ejecución de un programa de aplicación.

MODALIDADES OPERATIVAS DEL BASIC



PROCEDIMIENTO COMPLETO PARA LA ESCRITURA Y ACTIVACION DE UN PROGRAMA DE APLICACION



Estructura de las instrucciones Basic

Un programa en Basic está constituido por una serie de líneas (instrucciones) que especifican, en secuencia lógica, las acciones a efectuar. La introducción de cada línea tiene lugar al final de su escritura pulsando la tecla CR, como para cualquier otra introducción.

Cada línea ha de estar precedida por un número que la distingue de las demás. Este número es la «etiqueta» (label) de la instrucción, y la ejecución del programa (salvo si intervienen instrucciones de «salto») se realiza desde el label menor al mayor.

La numeración puede partir de un valor cualquiera y avanzar con cualquier paso, que incluso puede no ser constante. Las únicas limitaciones son el valor máximo (normalmente, 32.767, por lo que no puede haber una línea de número 40.000) y el orden progresivo. Si se introducen las instrucciones número 10, 35, 160, 2.000 y luego la número 20, el intérprete Basic coloca esta última línea en su lugar en la sucesión creciente (que no es la de introducción), y el programa estará constituido por la secuencia: 10, 20, 35, 160, 2.000, aunque la línea 20 haya sido la última en introducirse.

El formato de las instrucciones es el siguiente:

nnnn	**Instrucción**
1260	**PRINT 6 + 4 (CR)**

El símbolo nnnn es el número de línea; (CR) indica que para terminar e introducir la línea hay que pulsar la tecla CR.

En la misma línea física se pueden escribir varias líneas de programa (el número máximo de caracteres que se pueden escribir en una línea varía entre 72 y 255, según las versiones del Basic), separando las distintas instrucciones con el símbolo adecuado. El más usado es «:», aunque algunas máquinas utilizan el símbolo «back-slash» (\\). Por ejemplo, el programa:

```
10 PRINT  6 + 4
20 PRINT 10 - 2
30 PRINT  8 + 3
```

puede escribirse de la forma siguiente:

```
10 PRINT 6 + 4: PRINT 10 - 2: PRINT 8 + 3
```

Si la máquina adopta la otra simbología (\\), tendremos el siguiente programa:

```
10 PRINT 6 + 4 \ PRINT 10 -
   - 2 \ PRINT 8 + 3
```

Funciones de control

Como se ha dicho, en el teclado hay algunas teclas especiales que realizan ciertas funciones. Una de ellas es la tecla CONTROL, normalmente designada con las siglas CTRL. Esta tecla,

Códigos de control del Basic 80

CTRL + A	La máquina entra en la modalidad editor (corrección de programas).
CTRL + C	Interrumpe la ejecución de un programa en curso y pone la máquina en estado de instrucción.
CTRL + G	Sirve para activar la señal acústica de la que normalmente dispone la unidad de vídeo. Esta función se usa para llamar la atención del operador cuando se requiere su intervención. En este caso es el programa de aplicación el que ha de mandar al terminal el mismo código que se genera al presionar simultáneamente las teclas CTRL + G.
CTRL + H	Desplaza el cursor hacia atrás (hacia la izquierda) una posición y borra el último carácter. Por ejemplo, si en la pantalla aparece la palabra CASA, la instrucción CTRL + H la deja en CAS. Esta instrucción es útil en fase de corrección; introduciéndola varias veces, se borran varios caracteres consecutivos.
CTRL + I	Activa la tabulación; el cursor se desplaza 8 columnas por cada instrucción CTRL + I. La pantalla tiene normalmente 80 columnas, por lo que se pueden efectuar 10 tabulaciones como máximo.
CTRL + O	Detiene momentáneamente el output de un programa de aplicación en curso (por ejemplo, las impresiones); una segunda introducción de esta misma instrucción reactiva la salida previamente bloqueada (el programa sigue su curso).
CTRL + R	Reescribe una línea (se usa para duplicar líneas).
CTRL + S	Suspende la ejecución de un programa (no es equivalente a CTRL + C, que interrumpe definitivamente el programa).
CTRL + Q	Reactiva el programa suspendido momentáneamente con la anterior.
CTRL + U	Borra una línea.

usada junto con algunas letras (A, C, G, H, I, O, R, S, Q, U), da lugar a una secuencia de instrucciones que son interpretadas y efectuadas por el Basic. Los controles reconocidos en Basic se enumeran en la pág. 322. La simbología CTRL + letra indica que para activar la función hay que pulsar simultáneamente la tecla CTRL y la de la letra correspondiente.

Uso inmediato del Basic

El lenguaje Basic, gracias a la característica de tener un intérprete, permite realizar cálculos de forma «inmediata», sin más instrucciones que la introducción por teclado del cálculo a efectuar.
En los demás lenguajes no es posible esta modalidad de funcionamiento, puesto que antes hay que introducir la expresión desarrollada del cálculo a efectuar, como si se tratara de una línea «fuente» (es decir, utilizando una simbología de alto nivel, como Fortran, Cobol, etc.) y luego compilar la línea, como si se tratara de un programa cualquiera, y por último hacerla ejecutar. La característica fundamental del Basic es la de poseer un programa intérprete que analiza y traduce en código máquina cada línea en el momento de su introducción. El uso de esta posibilidad es sumamente útil en la búsqueda de errores.
Durante la fase de prueba de un programa es normal encontrar algunos errores (resultados inexactos de cálculos, presentación de datos erróneos, etc.).
Para determinar en qué punto se produce el error, se pueden efectuar los cálculos de uno en uno de forma «inmediata».
El usuario puede seguir el desarrollo del programa simulando sus funciones en el teclado y procediendo instrucción a instrucción; se puede, así, comprobar los resultados y encontrar la instrucción errónea.
Las operaciones que cabe efectuar de forma inmediata son las mismas del funcionamiento normal (programación), y se pueden usar los paréntesis. El cálculo a efectuar ha de ir precedido por una instrucción que especifique la función requerida. Si deseamos tener el resultado en pantalla, la instrucción es:

PRINT expresión a calcular (CR)

Mientras que en impresora es:

LPRINT expresión a calcular (CR)

Recordemos que el símbolo CR indica que para activar el cálculo hay que pulsar la tecla CR al final de su planteamiento. Por ejemplo, la instrucción

PRINT (7 + 3) / 2 (CR)

da en pantalla el resultado 4.
En funcionamiento inmediato se pueden usar también los operadores lógicos (AND, NOT, OR, etc.). Por ejemplo, PRINT 5 OR 2 da 7 (en binario, 5 = 101 y 2 = 10, por lo que 101 OR 10 = 111 = 7 decimal).
El lenguaje Basic (como todos los demás) dispone de un «repertorio» de funciones, o sea de un grupo de programas que realizan funciones matemáticas y que pueden utilizarse también de modo inmediato. Por ejemplo, la raíz cuadrada de un número es efectuada por uno de estos programas; el método con que se calcula la función, en este caso la raíz cuadrada, se denomina **algoritmo**.

Simbología, significado y prioridad de los operadores

Los operadores reconocidos en Basic son de cuatro clases:

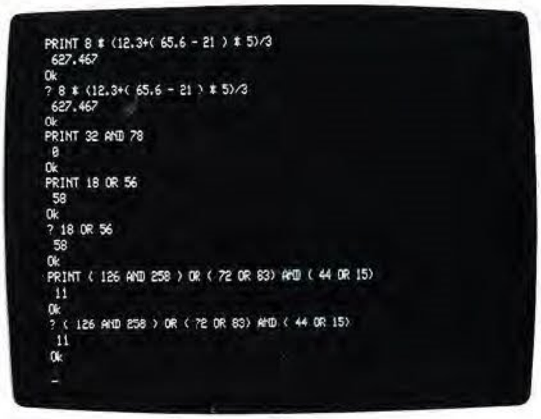
- **Aritméticos**
- **Relacionales**
- **Lógicos**
- **Funcionales**

Cada grupo tiene su propia «jerarquía», es decir, un orden de prioridad según el cual se efectúan los cálculos. El orden jerárquico puede variarse usando los paréntesis; el desarrollo comienza por los más internos y termina por los más externos, como en el cálculo normal de una expresión algebraica.
En un cálculo mixto, es decir, con distintos tipos de operadores, la prioridad es la indicada en la enumeración anterior. Por ejemplo, si un ordenador contiene operadores lógicos y operadores aritméticos, serán considerados antes los aritméticos y luego los lógicos.

Operadores aritméticos

Los operadores aritméticos previstos en Basic, por orden de prioridad, son los siguientes:

USO DEL BASIC EN MODALIDAD INMEDIATA




```
PRINT 8 * (12.3+( 65.6 - 21 ) * 5)/3
 627.467
Ok
? 8 * (12.3+( 65.6 - 21 ) * 5)/3
 627.467
Ok
PRINT 32 AND 78
 0
Ok
PRINT 18 OR 56
 58
Ok
? 18 OR 56
 58
Ok
PRINT ( 126 AND 258 ) OR ( 72 OR 83) AND ( 44 OR 15)
 11
Ok
? ( 126 AND 258 ) OR ( 72 OR 83) AND ( 44 OR 15)
 11
Ok

 _
```

Las operaciones aritméticas y lógicas pueden realizarse de forma inmediata utilizando una sola instrucción PRINT seguida de la expresión a calcular. Una vez introducida la expresión, debe pulsarse la tecla RETURN.

El sistema responde con el resultado de la operación que se le ha propuesto, y se prepara para recibir un nuevo input.

La instrucción PRINT puede sustituirse por el símbolo «?» el cual es interpretado con el mismo significado.

Como en la modalidad inmediata pueden utilizarse todas las funciones del Basic, el microordenador se convierte en una potentísima calculadora.

Operación	Símbolo	Ejemplo
Potencia	∧	3∧2
Cambio de signo	–	–3
Multiplicación	*	5*2
División	/	8 / 3
Suma	+	2 + 7
Resta	–	6 – 4

En algunas máquinas, el exponente se indica con el símbolo ↑, o con dos símbolos de multiplicación **. Por ejemplo, la operación 3^2 puede tener las simbologías: 3∧2, 3 ↑ 2, 3**2.

En el planteamiento de las expresiones a calcular, sea en modo inmediato o en programación, hay que prestar especial atención a la prioridad de los cálculos. Las operaciones se realizan en el orden dado en la tabla, luego el primer cálculo a efectuar es la elevación a potencia; siguen el cambio de signo, la multiplicación, etc. Este orden puede ser modificado mediante el uso de paréntesis.

Por ejemplo, la instrucción:

```
PRINT 5∧2 + 3*7/2
```

EJEMPLOS DE DESARROLLO DE LOS CALCULOS EN BASIC

	5∧2 + 3 * 7 / 2		5∧(2 + 3) * 7 / 2
Paso 1	25 + 3 * 7 / 2	**Paso 1**	5∧5 * 7 / 2
Paso 2	25 + 21 / 2	**Paso 2**	3125 * 7 / 2
Paso 3	25 + 10.5	**Paso 3**	21875 / 2
Paso 4	35.5	**Paso 4**	10937.5
	5∧2 + 3 * 7 / 2 = 35.5		5∧(2 + 3) * 7 / 2 = 10937.5

El primer cálculo efectuado es 5∧2 = 5^2 = 25, puesto que el exponente tiene prioridad máxima; sigue el operador * (multiplicación), luego el operador / (división) y por último el + (suma)

En este caso se efectúa antes el cálculo contenido entre paréntesis (5∧5 = 5^5 = 3125)

Expresión algebraica	Expresión en Basic	
$(5\times 3)-7:2$	(5 * 3) – 7 / 2	
$(9^2+2\times 6):4^3$	(9∧2 + 2 * 6) / 4∧3	(9^2 = 9∧2; 4^3 = 4∧3)
$\frac{3^2+4}{6}$	(3∧2 + 4) / 6	
$7^{2/3}+9$	7∧(2 / 3) + 9	
$\frac{5\times 3+2-1}{8^2}$	(5 * 3 + 2 – 1) / 8∧2	El cálculo puede indicarse mediante símbolos (A, B, C, etc.) a los que se asignará un valor numérico
$\frac{A\times B}{C}+C^2$	(A * B) / C + C∧2	

da 35.5*. En el gráfico de la pág. 325 se muestra la secuencia de los cálculos.
Modificando las prioridades mediante paréntesis, poniendo por ejemplo:

PRINT 5Λ(2 + 3) * 7/2

el resultado sería 10937.5. Para desarrollar un cálculo cualquiera, basta con sustituir los operadores aritméticos (suma, producto, etc.) por los correspondientes símbolos. En el gráfico de la pág. 325 se muestran algunos ejemplos.

Operadores relacionales

Se utilizan para comparar dos magnitudes. El resultado puede tener los valores 0, si la condición es falsa, y –1 si es cierta. Los operadores relacionales se usan normalmente en instrucciones de «decisión»



y el resultado de la operación (verdadero o falso) es gestionado por la instrucción misma; el usuario lo utiliza implícitamente. A continuación se enumeran, por orden de prioridad, los operadores relacionales del Basic.

Relación	Operador		Ejemplo
Igualdad	=	A = B	iguala el valor de A al de B, o bien la condición es verdadera si A es igual a B
Desigualdad	< >	A< >B	la condición es verdadera si A es distinto de B
Menor	<	A<B	la condición es verdadera si A es menor que B
Mayor	>	A>B	verdadero para A mayor que B
Menor o igual	< =	A<=B	verdadero tanto si A es menor que B como si es igual que B
Mayor o igual	> =	A>=B	verdadero tanto si A es mayor que B como si es igual a B

Un ejemplo inmediato del empleo de estos operadores lo tenemos en los programas de selección de datos. La selección dicotómica en archivos ordenados requiere, para determinar si el valor buscado se halla a la derecha o a la izquierda del que se ha leído, una serie de comparaciones efectuada mediante los operadores relacionales > (mayor que), < (menor que), = (igual a). En el gráfico superior de la pág. 327 se ve el diagrama de un paso durante la búsqueda binaria (dicotómica) de los datos.
Los operadores relacionales pueden incluirse en expresiones que contengan también operadores aritméticos (Λ, *, /, +, –); pero estos últimos tienen prioridad. Por ejemplo, en la expresión 3 + 5>7*2 primero se efectúan los cálculos aritméticos (3 + 5 y 7*2), y luego se toma en consideración el operador relacional. La expresión equivale, pues, a 8>14.

Operadores lógicos

Los operadores lógicos del Basic son los mismos del apartado de la pág. 73. La prioridad con la que son ejecutados es la siguiente:

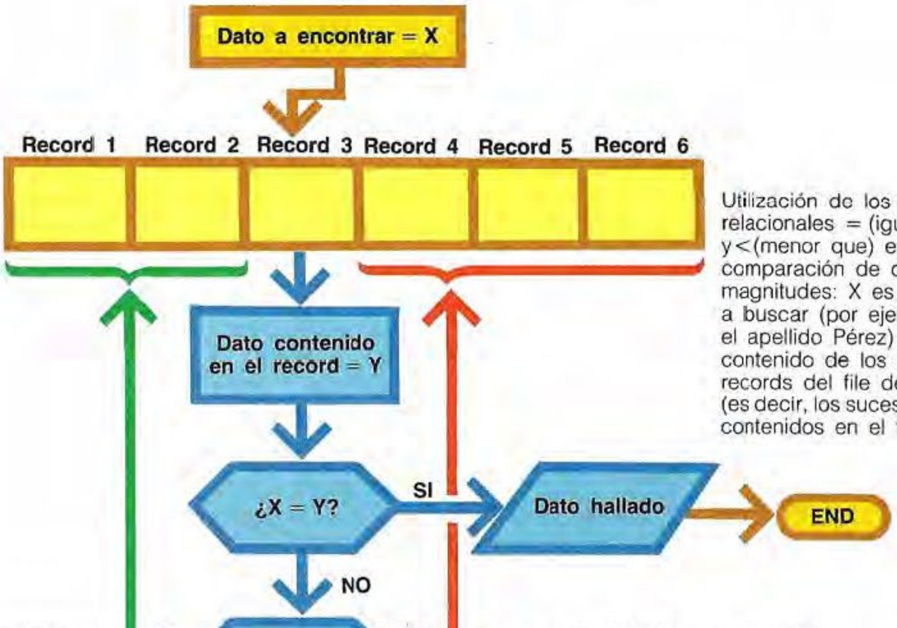
Operador	Datos	Resultado
NOT	A=1	NOT A=0
AND	A=1, B=0	A AND B=1 AND 0=0
OR	A=1, B=0	A OR B=1 OR 0=1
XOR	A=1, B=0	A XOR B=1 XOR 0=1

El Basic 80 contempla otros dos operadores: IMP y EQV. (Sus tablas de verdad se pueden ver en el gráfico contiguo). Pero no están en todas las versiones del Basic, y en los programas de aplicación se utilizan muy raramente.
Los operadores lógicos pueden trabajar con números enteros (los que la máquina puede contener en memoria están comprendidos entre –32768 y 32767) y operan sobre los bits que componen los números.
Los operadores aritméticos, relacionales y lógicos pueden utilizarse en la misma expresión.
Por ejemplo, la expresión:

(A + B) > C AND F = B

* De ahora en adelante se adoptará la representación de los números reales con punto decimal en lugar de coma. En el cálculo automático ésta es la representación estándar; por tanto, 35.5 en lugar de 35,5.

APLICACION DE OPERADORES RELACIONALES A LA BUSQUEDA DE DATOS



Utilización de los operadores relacionales = (igual a) y<(menor que) en la comparación de dos magnitudes: X es el elemento a buscar (por ejemplo, el apellido Pérez) e Y es el contenido de los distintos records del file de datos (es decir, los sucesivos apellidos contenidos en el file).

La condición X<Y es verdadera; el valor a encontrar (X) es menor que el dato examinado, y la búsqueda se ha de desplazar a la izquierda

La condición X<Y no se cumple; el valor a encontrar (X) es mayor que el dato examinado, por lo que la búsqueda se ha de desplazar hacia la derecha

TABLA DE VERDAD DE LOS OPERADORES IMP Y EQV

Operador IMP

		A IMP B
1	1	1
1	0	0
0	1	1
0	0	1

El operador IMP reconoce la secuencia 1 0 y suministra 0 en salida; en todos los demás casos la salida (resultado) es 1

Operador EQV

A	B	A EQV B
1	1	1
1	0	0
0	1	0
0	0	1

El operador EQV reconoce la situación de igualdad entre las dos entradas (ambas 1 o 0); en este caso suministra 1, de lo contrario 0

Modelado por ordenador

Antes de que una nueva máquina se ponga a la venta, ya sea una plataforma para perforaciones petrolíferas, un motor de automóvil, o un circuito electrónico, el fabricante ha de saber con exactitud cómo se comportará al usarla, y como los proyectos cambian rápida y radicalmente, el control de su fiabilidad y eficacia se convierte en una cuestión tan importante como compleja. La comprobación de aparatos mediante métodos destructivos es cara, y también lo es darles una resistencia elevadísima empleando más materiales de los necesarios. Afortunadamente, para solucionar este tipo de problemas se cuenta hoy con la ayuda de los ordenadores.

Los nuevos métodos se basan en modelos matemáticos que reproducen el comportamiento del aparato a examen. Si los cálculos evidencian algún punto débil en la estructura, el aparato puede proyectarse de nuevo en la mesa de diseño y, si es necesario, puede repetirse el proceso hasta que los cálculos demuestren que el funcionamiento es correcto. Sólo entonces se pasará a la fabricación; si todo ha sido bien calculado y construido, el aparato funcionará de la forma prevista.

Cuando se trata de maquinarias especialmente complejas, como una torre de perforaciones petrolíferas o un avión, los modelos matemáticos que las simulan también resultan muy complicados, y los cálculos que comportan son demasiado complejos para que pueda realizarlos el hombre. Entonces, la larga y tediosa elaboración matemática se confía al ordenador, y los proyectistas pueden dedicarse al trabajo creativo de proyectar los modelos matemáticos, controlando su exactitud inicial y buscando la forma de mejorarlos hasta que funcionen de la forma deseada.

Esta aplicación de los ordenadores, centrada en la realización de un modelo del comportamiento de una estructura, se denomina «simulación por ordenador», y forma parte de un más amplio campo de utilización de la electrónica, en el que el ordenador mismo diseña y modifica los detalles de un proyecto, bien en las representaciones normales en dos dimensiones, bien en la representación tridimensional. Los métodos del modelado computerizado se dividen en dos grupos principales: la elaboración numérica de las ecuaciones matemáticas que describen el comportamiento físico de la estructura, y la visualización de los resultados del cálculo, que permite al proyectista corregir los errores.

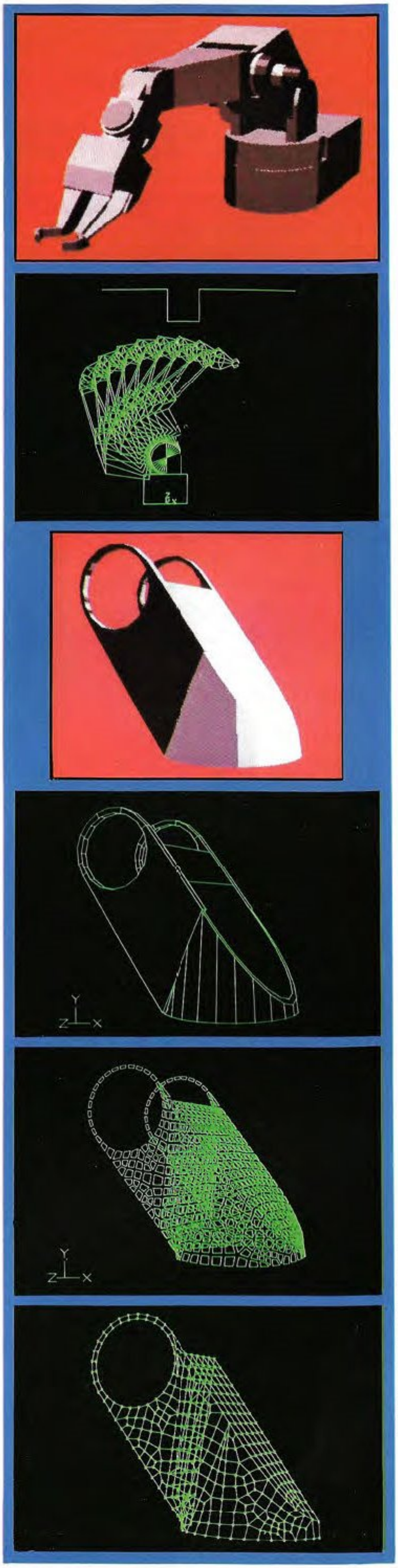
Los resultados de la elaboración numérica han de ser comunicados al usuario de forma que pueda comprenderlos casi a simple vista y, a tal objeto, el tipo de comunicación más accesible para un técnico suele ser un dibujo o un diagrama más bien que un conjunto de números. Por eso en el modelado computerizado se utilizan ampliamente los plotter *(aparatos electromecánicos que realizan dibujos a partir de instrucciones del ordenador) y los monitores (pantallas similares a las de televisión en las que aparece el diagrama elaborado por el ordenador). Ambos medios entran en la categoría de los denominados sistemas gráficos.*

Cuando el terminal de salida de un ordenador incluye un plotter, se puede disponer de una grabación permanente del diagrama o el dibujo producido por el ordenador. El documento se denomina hard copy. *A menudo el plotter es indispensable, porque el diagrama en cuestión es demasiado complejo para la respuesta relativamente somera que puede dar el monitor. Por otra parte, una imagen puede ser dibujada mucho más rápidamente en la pantalla que en el plotter, y con la misma rapidez puede modificarse para mostrar el objeto desde ángulos distintos. Además, el desarrollo del proceso gráfico puede ser controlado durante todas sus fases.*

El diseño de circuitos electrónicos es uno de los campos en que el ordenador es de mayor utilidad. Los circuitos digitales constan de miles de «puertas lógicas» electrónicas que controlan minúsculos impulsos de tensión, permitiendo o impidiendo a la corriente eléctrica que pase a través de las diversas partes del circuito, según la presencia o ausencia de otros impulsos de tensión. Los circuitos pueden ser extremadamente complejos, y el proyectista que descuidara el funcionamiento de una sola de las innumerables puertas, tras construir laboriosamente el circuito, se encontraría con que no funciona como estaba previsto. Un ordenador puede manipular con enorme facilidad los miles de elementos sin errores ni descuidos.

También la previsión del comportamiento de estructuras mecánicas en funcionamiento se facilita con los modelos de ordenador. Resultaría extremadamente caro, por ejemplo, construir torres de perforación auténticas para comprobar cómo responden, del mismo modo que sería prácticamente imposible someterlas a todas las

situaciones de carga que pueden darse.
Entre los instrumentos matemáticos utilizados para el análisis de las tensiones está el método de los elementos finitos, que permite estudiar el comportamiento mecánico de cualquier estructura, desde el armazón de un aeroplano hasta un simple diente de engranaje, esquematizándola mediante un conjunto de bloques elementales, como triángulos, rectángulos o barras.
El ingeniero introduce en el ordenador los datos que describen el modelo construido con los elementos finitos de la estructura que ha de ser proyectada. El modelo mismo incluirá como datos los tipos de elementos a usar y el modo en que han de ser dispuestos, la situación de los nudos y los materiales a utilizar. De este modo, el ordenador puede realizar un dibujo del modelo de elementos finitos, que aparecerá como una telaraña de líneas que se entrecruzan conectando los distintos puntos y planos, reproduciendo la forma global de la estructura real. El proyectista puede pedir al ordenador que haga girar el modelo para examinarlo desde otro ángulo, o que amplíe un elemento concreto para estudiarlo con detalle. Además, el proyectista puede establecer una situación de carga, decidiendo cuáles han de ser las presiones a aplicar a las distintas partes del modelo. Las deformaciones resultantes aparecerán en la pantalla del ordenador, eventualmente superpuestas a la imagen de la estructura no deformada, de manera que el proyectista pueda ver inmediatamente las alteraciones producidas.
El proyectista también puede analizar los efectos producidos en la estructura por condiciones de funcionamiento anómalas o por situaciones accidentales; ver, por ejemplo, qué le pasaría a una plataforma petrolífera si perdiera una columna de sostén en un mar tormentoso.
El trabajo matemático y de elaboración sólo es útil, evidentemente, si las previsiones resultan exactas en la realidad. Una manera de garantizar una mayor precisión consiste en utilizar elementos finitos muy pequeños, que en la pantalla aparecen delimitados por una red finísima. La utilización de una red fina, en vez de una malla ancha, produce, sin embargo, un aumento de los costos de elaboración. El proyectista em-



SDRC Engineering Services Ltd.

En las dos primeras imágenes: visualización en monitor de un brazo articulado y una secuencia de sus movimientos. En las demás: pieza de fabricación y retículos de elementos finitos.

pleará más tiempo en introducir los elementos en el ordenador y en controlarlos, y también el ordenador tardará más en realizar el análisis y dar los resultados. Por otra parte, en algunas aplicaciones, una red de malla ancha también puede dar resultados válidos.

*Los modelos por ordenador se revelan especialmente útiles para prever el comportamiento de objetos en movimiento. En este sentido, un ejemplo espectacular lo constituye la puesta en funcionamiento de una torre de perforaciones petrolíferas (*derrick*), que tiene una estructura reticular extremadamente compleja. La torre se transporta al lugar de instalación, tumbada sobre un costado, en una embarcación especial, y una vez allí se sumerge en el agua y se hace girar 90° para que se ponga sobre el fondo marino en el punto establecido.*

El resultado gráfico de la simulación es, en este caso, un conjunto de dibujos que reproducen la torre en determinados momentos. Los dibujos pueden superponerse, de forma que el proyectista tenga una clara visión de cómo se mueve la torre. Los dibujos son tridimensionales, por lo que se puede tener a voluntad una vista lateral o desde arriba. Puesto que a menudo es importante saber qué partes de la estructura se encuentran bajo el agua en un momento cualquiera, el plotter puede realizar un dibujo con una máscara superpuesta que indica constantemente el nivel del agua.

*Los modelos por ordenador pueden usarse también para estudiar el comportamiento mecánico del cuerpo humano y, especialmente, cómo se mueve el hombre al manejar una máquina. Uno de estos programas es el SAMMIE (*System for Aiding Man-Machine Interaction Evaluation*: sistema para la evaluación de la interacción hombre-máquina), con el que se dibujan figuras humanas en tres dimensiones, superpuestas a imágenes, también tridimensionales, de maquinarias y edificios. Las figuras pueden moverse, sentarse o levantarse, y así se puede comprobar, por ejemplo, si alcanzan fácilmente los mandos de una máquina.*

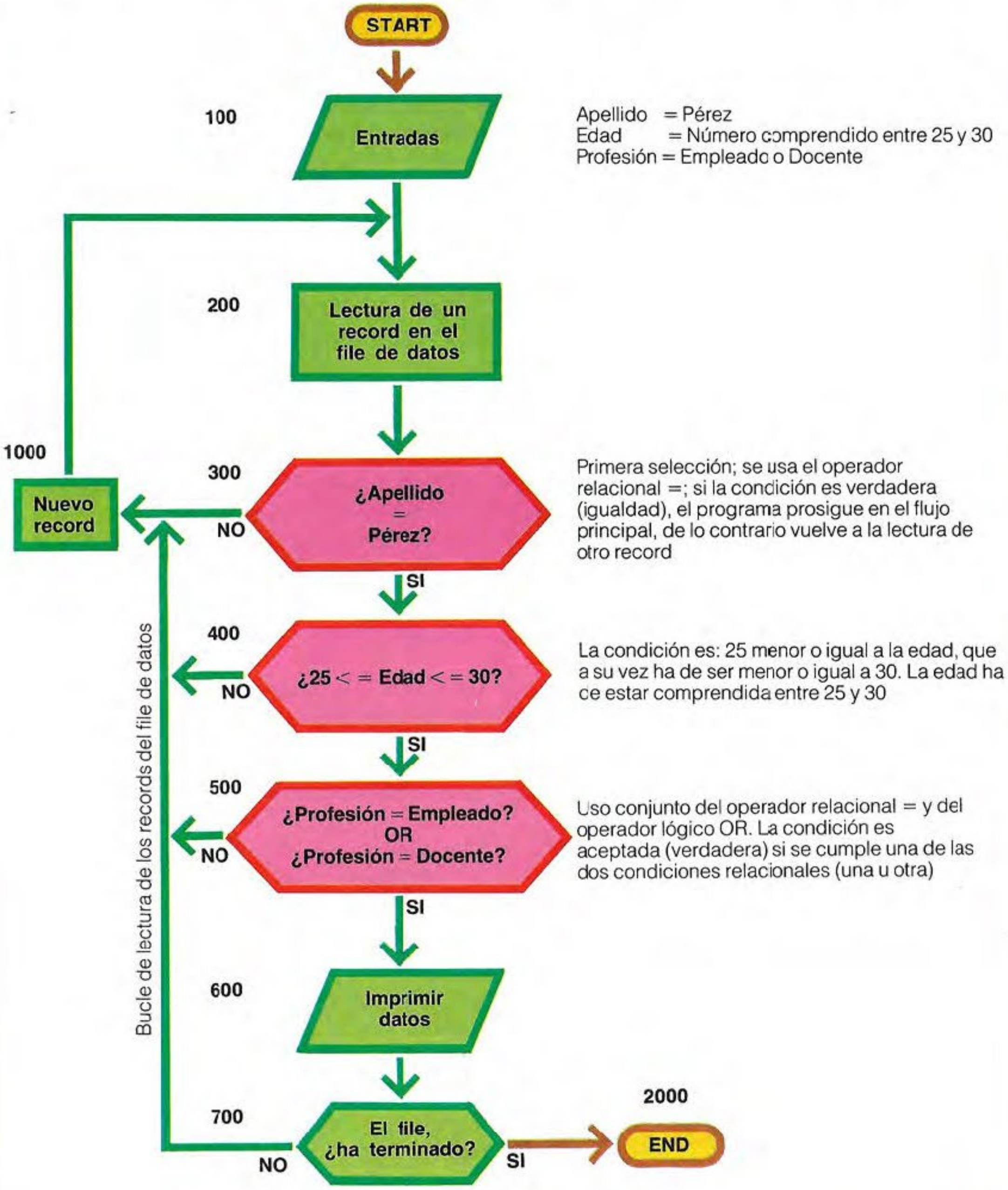
El modelo (o maniquí) está hecho de forma que se mueva exactamente como un ser humano, flexionando correctamente todas las articulaciones, girando la cabeza, etc. Si un determinado movimiento impuesto al maniquí no es natural, por ejemplo, si el brazo se ha de girar demasiado o en sentido erróneo, suena una señal de alarma. Como las personas tienen diferentes estaturas, también la del maniquí se puede variar. El ordenador está programado para representar personas de estatura media, pero también puede reproducir figuras humanas muy bajas, delgadas, altas o gruesas. El paso siguiente consiste en diseñar el ambiente en el que ha de moverse el maniquí, por ejemplo, la cabina de conducción de un tractor, una oficina de paredes móviles, o una cocina con sus instalaciones. Mediante una serie de sencillas instrucciones, el proyectista dice al ordenador cómo ha de ser el lugar de trabajo, y el ordenador lo dibuja esquemáticamente, insertando en él al maniquí en la posición que se desee.

Los dibujos creados por el SAMMIE son tridimensionales, el programador puede cambiar el punto de vista o acercarse a una parte del modelo. El técnico puede girar alrededor o incluso entrar en la situación de trabajo para poder comprobar que el maniquí tenga suficiente sitio. Por ejemplo, en la cabina de un tractor, el asiento ha de estar lo suficientemente atrás como para que las piernas del conductor quepan debajo del volante. El proyectista puede establecer una determinada disposición de los mandos, y si alguno de ellos quedara fuera del alcance del maniquí, el ordenador lo comunicaría mediante un mensaje impreso. Entonces el proyectista puede modificar el modelo y pedir al ordenador que repita la prueba.

Otra ventaja del SAMMIE es que permite al proyectista darse cuenta de lo que ve el maniquí, y de cómo cambia su visual al mover la cabeza y los ojos.

El empleo de modelos computerizados puede permitir ahorrar las grandes sumas destinadas hasta ahora a la realización de prototipos, así como el tiempo antes necesario para la evaluación de los resultados de pruebas complejas. El empleo del ordenador puede ser de gran ayuda para alcanzar elevados estándares de seguridad, puesto que permite a los proyectistas comprobar fácil y rápidamente el comportamiento del objeto a examen en todas las condiciones, tanto normales como anormales. A medida que los ordenadores se vuelvan más rápidos y potentes, se podrán utilizar sistemas gráficos mejores, capaces de dar al proyectista una representación visual más realista de lo que sucede, y que permitirán, asimismo, el control «en tiempo real» de los objetos en movimiento para poder investigar con mayor profundidad su comportamiento en la práctica.

USO CONJUNTO DE OPERADORES RELACIONALES Y LOGICOS



es verdadera (da como resultado –1) si la suma A + B es mayor que C y, simultáneamente, el valor de F es igual al de B. En el gráfico de esta página se muestra un ejemplo de la utilización conjunta de varios tipos de operadores. El problema al que se refiere el ejemplo es la impresión estadística del contenido de un file de direcciones. El file contiene datos personales. Se desea la impresión de los datos de todas las personas que se apellidan Pérez, de edad comprendida entre 25 y 30 años (ambos inclusive) y que sean empleados o docentes.

Operadores funcionales

Una secuencia de cálculo puede convertirse en una «función», representada por la serie de operaciones a efectuar con un determinado operando. Así, el siguiente cálculo del importe correspondiente al 18% de una cantidad:

$$\text{Importe} = \frac{\text{Cantidad} \times 18}{100} = \text{Cantidad} \times 0.18$$

puede convertirse en una función consistente en multiplicar por 0.18 la cantidad.
En términos matemáticos, la función se indica de la forma siguiente:

$$\text{Importe} = f\,(\text{Cantidad})$$

La expresión f (Cantidad) (función de la Cantidad) indica, en este caso, el producto de Cantidad × 0.18. Una simbología muy similar es la utilizada en Basic. El símbolo de función lo constituyen las letras FN, y el nombre de la función es una tercera letra. Por ejemplo, FNA se refiere a una función (FN) de nombre A. En el ejemplo anterior el operando es «Cantidad», y la forma completa de la función es:

FNA (Cantidad)

donde FNA (Cantidad) indica el producto Cantidad × 0.18.
Una vez definida la función, puede utilizarse en un cálculo cualquiera. Por ejemplo, si el 18% fuera un descuento, para calcular el precio tendríamos antes que calcular el descuento y luego restarlo de la cantidad:

$$\text{Descuento} = \frac{\text{Cantidad} \times 18}{100} = \text{Cantidad} \times 0.18$$

(siendo Cantidad el precio sin descuento)

$$\text{Precio} = \text{Cantidad} - \text{Descuento} = \\ = \text{Cantidad} - \text{Cantidad} \times 0.18$$

Puesto que FNA (Cantidad) determina el importe del descuento, tendremos:

Definición
Descuento = FNA (Cantidad) (es decir, Cantidad × 0.18)

Cálculo
Precio = Cantidad – FNA (Cantidad)

Las funciones de este tipo se denominan «funciones definidas por el usuario», puesto que no están definidas a priori en el Basic y, por tanto, han de ser definidas por el programador. La instrucción de «definición» de una función ha de darse antes de la utilización de la propia función, de lo contrario el sistema emite un diagnóstico de función desconocida y se detiene en espera de correcciones. Para informar al sistema de la existencia de la nueva función FNA (Cantidad), es necesaria la instrucción de definición DEF FNA (es decir, definir la función A). Como en casi todas las instrucciones Basic, la simbología indica el nombre mismo de la operación a realizar. La operación es la DEFINICION, el código en Basic es DEF, y deriva de las primeras letras de la palabra «definición».
Para definir una función, por ejemplo la FNA (Cantidad) antes citada, la instrucción es:

DEF FNA (Cantidad) = Cantidad * 0.18

- Código de la instrucción de definición
- Símbolo que indica una función a la máquina
- Nombre de la función
- Parámetro
- Cálculo realizado por la función

Las funciones definidas por el usuario pueden ser de tres clases:

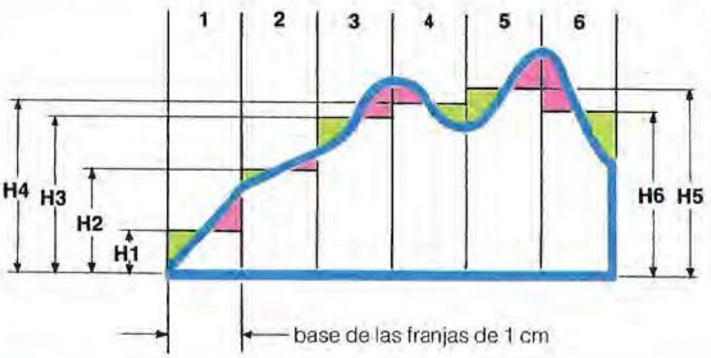
— **reales**: todas las que operan con números reales, es decir, compuestos de una parte entera y otra decimal;
— **enteras**: las que operan sólo con números enteros;
— **de cadena**: las que operan con caracteres (letras o números).

Las características de las tres clases y la forma de definir las funciones en los tres casos se expondrán más adelante.
Además de las funciones definidas por el usuario, hay funciones «de repertorio», ya preparadas, que efectúan los cálculos matemáticos más corrientes. Estas funciones, definidas a nivel de sistema, son «llamadas» (o sea utilizadas) con un nombre simbólico (definido en el Basic) que ya no utiliza la sigla FNX (X = nombre de la función). Por ejemplo, la función que calcula la raíz cuadrada tiene por nombre la sigla SQR (abreviatura de square root: raíz cuadrada); así, para extraer la raíz cuadrada del número 1673 habrá que escribir:

SQR (1673)
código parámetro

Para comprender por qué en los ordenadores hace falta una función (en vez de un cálculo directo) para sacar una raíz, hay que analizar la estructura interna de la máquina. El sistema efectúa los cálculos utilizando los microprogramas (microcódigos) contenidos en la ALU (unidad aritmética y lógica). En ella, las señales que representan los operandos (los números) y los operadores (las operaciones a efectuar) son interpretados y enviados a determinados circuitos que suministran el resultado en la salida.
Sin embargo, estos circuitos sólo pueden realizar unas cuantas clases de cálculos con el método del complemento, en esencia, sumas y

PREPARACION GEOMETRICA PARA EL DESARROLLO DE UN CALCULO DE AREAS

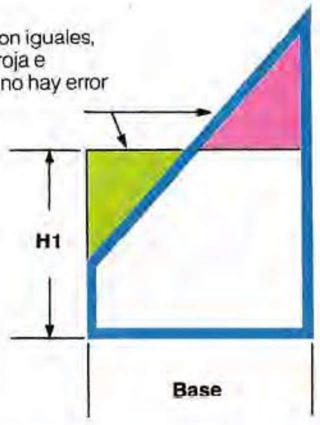


La figura plana (trazo azul) está dividida en un determinado número (6) de franjas verticales; el área total es la suma de las áreas parciales de las franjas, que se halla multiplicando la base por la altura correspondiente (H1, H2, etc.)

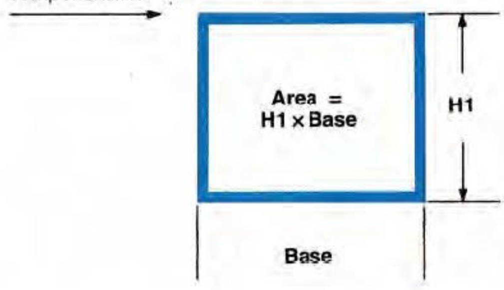
restas. Las operaciones de multiplicación y división son realizadas por microprogramas normalmente externos a la ALU.
Estas operaciones se denominan «realizadas por software», puesto que no implican directamente a los circuitos. Todos los demás cálculos, por ejemplo la raíz cuadrada, han de realizarse utilizando sólo las cuatro operaciones fundamentales. La raíz cuadrada tiene su propio algoritmo de cálculo, que se activa escribiendo la instrucción SQR (X), donde X indica el número genérico del que se desea sacar la raíz.
A título de ejemplo, apliquemos el concepto de función a un programa para el cálculo de áreas. Los términos del problema son los siguientes: dada una figura plana, delimitada por una recta horizontal, dos rectas verticales y una curva,

METODO DE COMPENSACION PARA EL CALCULO DE AREAS

Si estas dos áreas son iguales, eliminando la parte roja e incluyendo la verde no hay error



Con la compensación entre área incluida (roja) y área
excluida (verde), la figura equivale a un rectángulo con la misma base y altura igual a la de compensación



calcular su área. El primer paso consiste en dividir la figura en un cierto número de franjas verticales (ver gráfico de pág. 333), todas de igual base, por ejemplo de 1 cm de longitud.
Cada franja puede considerarse un rectángulo de una determinada altura. La altura del rectángulo equivalente (H1 en el gráfico superior) se elige de forma que la zona excluida (zona roja) sea equivalente a la zona añadida (zona verde). De esta manera, aunque se excluye una parte de la figura (roja), se incluye una zona de igual área (verde) que en la figura no existía. En otras palabras, el área de cada franja vertical se obtiene como si se tratara de un rectángulo de base H1, H2, etc. La división en un cierto número de franjas es necesaria para minimizar el error que se cometería dibujando «a ojo» la recta de compensación sobre la anchura de toda la base. Una vez terminado el gráfico sobre el que pueden leerse las distintas alturas, se puede realizar el diagrama de flujo del programa. Las funciones a realizar son:

1 / toma del número de divisiones verticales (en general, seis no tienen por qué ser suficientes)

2 / toma de la medida de la base (en el ejemplo, 1 cm)

3 / bucle de toma de las alturas (en número igual al de franjas), cálculo de las áreas de los rectángulos y suma de las mismas (área total)

4 / presentación del resultado (suma de las áreas)

El cálculo más recurrente es el área de un rectángulo; para realizarlo rápidamente, podemos definir una función que opera con los parámetros base y altura.
El área de un rectángulo se obtiene multiplicando la base por la altura:

Área = Base × Altura

La función correspondiente (por ejemplo, de nombre A) es:

DEF FNA (Base, Altura) =
= Base × Altura

- Instrucción de definición
- Símbolo de función
- Nombre de la función
- Parámetros
- Cálculo

Ver el diagrama de flujo en la pág. 336.
Este método se denomina «integración gráfica de una función». La curva cuya área delimitada se desea conocer es la función (en el sentido matemático del término, que no hay que confundir con las funciones del Basic), y el cálculo del área es un método para el cálculo de la integral de la función en el intervalo correspondiente a la base. La integración es uno de los aspectos más complejos de las matemáticas, y ello hasta el punto de que el cálculo de una integral por medios analíticos puede no ser posible. En los cálculos científicos se usa el método expuesto, aunque de una forma que permite eliminar la parte gráfica a cargo del usuario.

TEST 9

1 / Escribir la instrucción que de manera "inmediata" permite calcular la media de los números: 5.2, 15.21, 7.43, de forma que el resultado aparezca en la pantalla y en la impresora.

2 / Escribir el resultado de las siguientes expresiones (atención a la prioridad de los operadores): 3 + 2 AND 3 × 4; 5 + (2 AND 3) × 4.

3 / Un file contiene el archivo de una biblioteca de 1000 volúmenes (1000 registros). Los datos están divididos en los siguientes campos:

1 / Título del libro	30 caracteres
2 / Tema	10 caracteres
3 / Especialidad	10 caracteres
4 / Idioma (siglas)	3 caracteres
5 / Disponibilidad	1 carácter

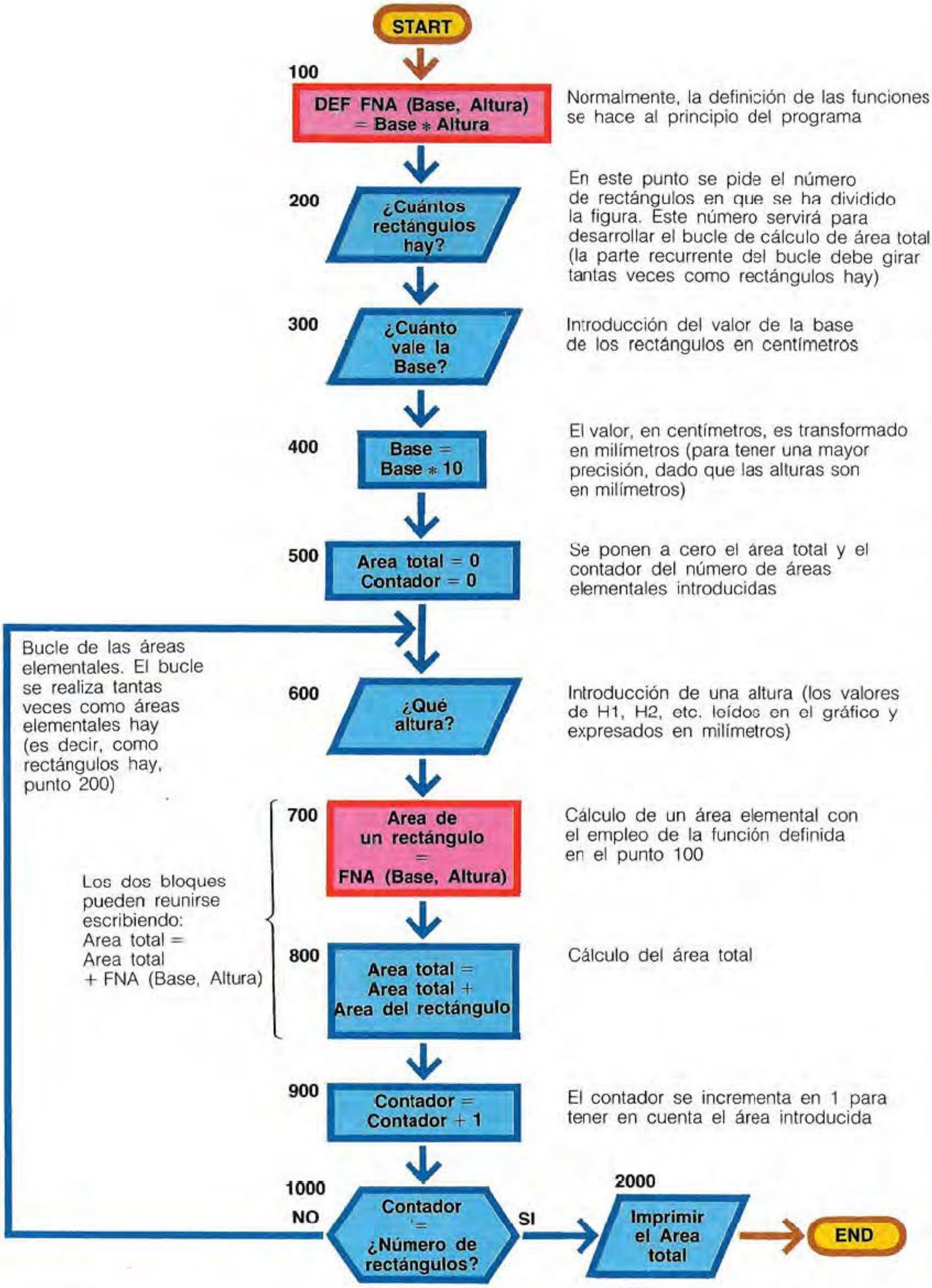
Los temas (campo 2) son una subdivisión a nivel general (por ejemplo, historia, geografía, etc.); la especialidad es la subdivisión del tema en sus partes (por ejemplo, para la historia las especialidades pueden ser: prehistoria, época romana, etc.). El campo 4 es el idioma en que está escrito el libro (ESP = español, FRA = francés, etc.) La disponibilidad (5) es un flag que puede valer 0 si el libro se ha prestado o 1 si se halla en la biblioteca.
Diseñar el diagrama de flujo de un programa que busque todos los libros presentes (no prestados) escritos en francés (FRA) y en inglés (ING) que hablen de poetas. El tema es Arte; la especialidad, Poetas.

4 / ¿Cuántos volúmenes del archivo del que se habla en la pregunta 3 puede contener un diskette de 1 Mbyte?

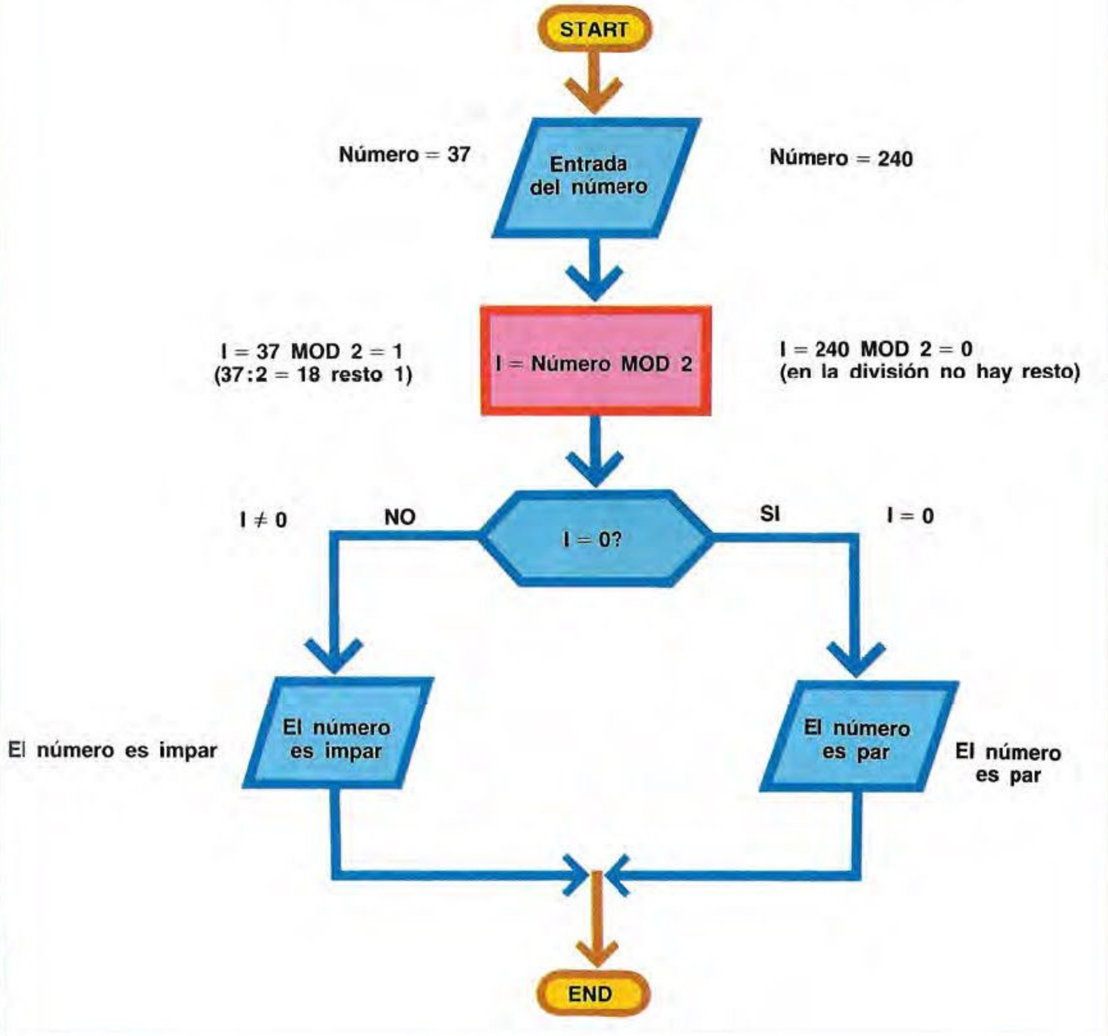
5 / Escribir la función para el cálculo de la hipotenusa de un triángulo rectángulo (usando la definición de funciones de usuario: DEF FN...).

Las soluciones, en las págs. 350 y 351.

DIAGRAMA DE FLUJO PARA EL CALCULO AREAS



APLICACION DEL OPERADOR MOD PARA DETERMINAR SI UN NUMERO ES PAR O IMPAR



Operador división entera

En la división entera, indicada con el símbolo \, los operandos se redondean como enteros antes de efectuar la división, cuyo resultado también se da como entero.
Por ejemplo:

35.7 \ 4.8 se evalúa como 36/5 = 7

Si la conversión de los operandos da un valor que excede los límites establecidos (– 32768, + 32767), hay error. Por ejemplo, el cálculo (75.4 * 56621) \ 5.6 no puede evaluarse, puesto que el numerador es mayor que 32767.

Operador módulo

El operador módulo (MOD) da un valor entero que es el resto de la división entre dos números. Por ejemplo:

7 MOD 2 = 1, puesto que 7:2 = 3 con resto 1
9 MOD 3 = 0, puesto que 9:3 = 3 con resto 0

El operador MOD es muy útil para comprobar la divisibilidad de un número por otro: si el primer número es divisible por el segundo, el resultado de MOD es 0; de lo contrario, es distinto de 0. En el gráfico superior se muestra el diagrama de flujo de un programa para comprobar si un número es par o impar.

Resumiendo:
El Basic tiene tres modalidades operativas:
— Estado de instrucción: para la introducción de las instrucciones.
— Ejecución: desarrollo de programas y de cálculos inmediatos.
— Editor: introducción y eventuales modificaciones de programas.
Los operadores reconocidos en Basic (por orden de prioridad) son:
— Aritméticos: operaciones matemáticas normales(∧, signo, *, /, +, −).
— Relacionales: igual, mayor, menor y sus combinaciones.
— Lógicos: los mismos operadores lógicos vistos en el capítulo correspondiente, más otros dos suplementarios (en algunas versiones).
— Funcionales: funciones definidas por el usuario (DEF FNX) o funciones de sistema.
— División entera y módulo: operadores \ y MOD.
El programador puede definir una función cualquiera utilizando la simbología: DEF FNX (X = nombre de la función, una letra cualquiera).
El Basic posee un amplio repertorio de funciones matemáticas ya establecidas. El programador puede utilizarlas llamándolas con su nombre simbólico.

El control se realiza comprobando la divisibilidad del número por 2 (si el número es divisible por 2, es par; de lo contrario, es impar).

Constantes y variables

En cualquier lenguaje simbólico, los datos a elaborar pueden ser de dos clases: **constantes** y **variables**.
Los constantes tienen valor fijo, mientras que los variables toman valores distintos durante la ejecución del programa.
Los variables se representan con nombres simbólicos; los constantes se indican directamente con su valor.

Por ejemplo, si se desea calcular la longitud de una circunferencia, la fórmula a utilizar es: Circunferencia = 2 × π × R. El símbolo π vale 3.14..., y es lo mismo indicar el cálculo con el símbolo π que con su valor explícito: Circunferencia = 2 × 3.14 × R. El símbolo π es una constante (numérica; como veremos, hay otro tipo de constante: la cadena; el símbolo R (radio de la circunferencia) es una variable, puesto que puede tomar un valor cualquiera (las cadenas también pueden ser variables).

Constantes numéricas

Las utilizadas en Basic son de cinco clases:
1 / Enteras (integer): representadas por un número entero (con signo) comprendido en el intervalo de valores −32768/+32767, seguido por el símbolo %.
2 / Reales de coma fija (fixed point): números reales (con decimales) negativos y positivos.*
3 / Reales de coma flotante (floating point): números reales en representación exponencial. La representación exponencial permite expresar un número utilizando potencias de 10. Por ejemplo, el número 1400 puede escribirse en la forma $14 \times 100 = 14 \times 10^2$. Análogamente, el 121.46 (o sea 121,46) es $12146/100 = 12146/10^2 = 12146 \times 10^{-2}$.
Recuérdese que 1/100 equivale a $1/10^2$, o sea 10^{-2}; el exponente 2, que en el denominador era positivo, se vuelve negativo en el numerador.
El formato que se utiliza en Basic para indicar un número de coma flotante es: número (mantisa), E (símbolo de «exponente»), exponente. Por ejemplo:

Valor numérico	Notación exponencial	Notación en Basic
1400	14×10^2	14 E 2
1400	1.4×10^3	1.4 E 3
1400	0.14×10^4	0.14 E 4
121.46	12146×10^{-2}	12146 E−2
121.46	1.2146×10^2	1.2146 E 2

Las distintas formas de representación de 1400 y 121.46 son equivalentes.
4 / Hexadecimales (HEX): son los números en representación hexadecimal. Para indicarle a la máquina que se va a usar la notación hexadecimal, el número ha de ir precedido por los símbolos &H. Por ejemplo, el número &H70 significa

* Aunque la notación habitual en los ordenadores para la coma decimal es el punto, se denomina coma.

112 decimal (70 hexadecimal es, en decimal, $7 \times 16 + 0 \times 16 = 7 \times 16 = 112$).

5 / Octales (OCT): son los números representados en octal; su símbolo es & («y» comercial). Por ejemplo, el número &70 tiene el valor decimal 56. Las notaciones octal y hexadecimal son útiles cuando se desea hacer referencia directa a una memoria, ya que el ordenador utiliza la simbología binaria, y éstas son las notaciones más próximas a ella. Por ejemplo, supongamos que queremos aislar el bit número 8 de un determinado dato. Esta función (ver código ASCII y transmisión de datos) puede obtenerse aplicando el operador AND entre el dato y una «máscara» constituida por un número binario compuesto todo de ceros menos el bit 8, de valor 1. Indicando con el nombre genérico de X el valor del dato, la función es: X AND 10000000. El valor numérico octal de esta máscara (10000000) es:

Valor	4 2 1	4 2 1	4 2 1
Binario	1 0	0 0 0	0 0 0
Octal	2	0	0

10 000 000 = 200 (octal)

A nivel de programación, el valor de la máscara se indicará en la forma &200.

Precisión

La representación interna de los números enteros utiliza 16 bits (2 bytes), luego la precisión de los números está limitada por el valor decimal que se puede escribir con 16 bits. Dicho valor se halla comprendido en el intervalo – 32768/+ 32767. En realidad, los bits utilizados para el valor numérico son 15, puesto que el bit 16 indica el signo del número.

El número es negativo si el bit 16 vale 1, de lo contrario (bit 16 = 0) es positivo. Recuérdese que para pasar de la representación binaria de un número negativo a su valor absoluto, hay que complementar sus bits y luego sumarle 1.

Por ejemplo, para obtener el valor del número binario 1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1 (negativo, puesto que el bit 16 vale 1), antes hay que complementar cada bit:

número	1 1 1 1 1 1 0 0 1 1 0 0 0 0 1 1
complemento	0 0 0 0 0 0 1 1 0 0 1 1 1 1 0 0
	+
sumar 1	1
	1 1 0 0 1 1 1 1 0 1

Obtenemos así: 1 1 0 0 1 1 1 1 0 1 = $2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^2 + 2^0 = 829$. Por lo tanto, el valor numérico es – 829.

Los números reales pueden memorizarse con simple precisión y con doble precisión. Con simple precisión se utilizan 32 bits, y el número puede variar entre 1×10^{-38} y 1×10^{38}, con siete cifras significativas.

Por ejemplo, el número $135.794397 \times 10^{25}$ queda, en realidad, truncado en 135.7943×10^{25}.

Las siete cifras significativas se utilizan en los cálculos, mientras que en la fase de impresión se toman sólo seis; por tanto, el número anterior, en fase de impresión, se convierte en 135.794 E 25 (recuérdese que E 25 significa $\times 10^{25}$).

Un número en simple precisión puede indicarse con el símbolo ! Así, 1.35 E 2 y 135! indican el mismo número (135) en simple precisión.

Para aumentar el número de cifras útiles y, por tanto, la precisión de los cálculos, se utiliza la doble precisión. Con este formato, para representar un número en la memoria se utilizan 64 bits. El valor del número puede variar entre 1×10^{-308} y 1×10^{308}, con dieciséis cifras significativas en los cálculos y quince en impresión.

La simbología para definir un número en doble precisión es la misma que se utiliza en simple precisión, sustituyendo la E por una D (inicial de la frase «double precision»). El número 135.794397 E 25 está en simple precisión (las dos últimas cifras, 9 y 7, se pierden); escribiendo 135.794397 D 25, el número está en doble precisión, y no se pierde ninguna cifra.

La razón por la que los números en simple y doble precisión son representados con una cifra menos que el formato de memorización, es el redondeo que el Basic efectúa de forma automática.

El símbolo D se usa para escribir un número en doble precisión en la forma exponencial; si el número no se expresa en esta forma, se utiliza el símbolo #. Por ejemplo, si se escribe 372.0, el número está en simple precisión; añadiendo el símbolo #, la precisión se hace el doble: 372.0 # está en doble precisión. Las dos notaciones, 372.0 y 372.0 #, en este caso concreto, son equivalentes, puesto que no hay cifras que se salgan de la simple precisión; sin embargo, si en el programa está previsto el uso del número en cálculos junto a valores en doble precisión, es conveniente usar una notación homogénea. En la tabla superior de la pág. 340 se enumeran las distintas formas de representación interna.

Precisión en la representación de los números

Tipo	Espacio en memoria	Valores límite	Cifras significativas	Cifras en presentación	Ejemplo
Enteros	2 bytes	$-32768 \div 32767$	5	5	21743
Simple precisión	4 bytes	$\pm 10^{-38} \div \pm 10^{+38}$	7	6	121.3E7
Doble precisión	8 bytes	$\pm 10^{-308} \div \pm 10^{+308}$	16	15	275.8D9 127.41 #
Hexadecimales	2 bytes	$0 \div FFFF$			&H57
Octales	2 bytes	$0 \div 177777$			&75

Aclararemos lo expuesto con algunos ejemplos:

Valor introducido	Significado
42745 %	Error. Un entero no puede ser mayor que 32767.
&12 * 5	El resultado es 50, puesto que el número octal &12 corresponde al 10 decimal.
&12 + &H12	Resultado: 28 (&12 = 10, &H12 = 18; el símbolo &H atribuye al número 12 el significado de número hexadecimal).
1357.92486E21	Las dos últimas cifras (8 y 6) se pierden, puesto que el número está en simple precisión.
1357.92486 #	Correcto. Todas las cifras se conservan, puesto que el símbolo # indica doble precisión.
13.5792486D2	Representa el valor anterior en notación exponencial.
579.3E41	Error. El exponente, en simple precisión, ha de estar comprendido entre –38 y +38.
579.3D41	Representación correcta del número anterior.
157.2!	El símbolo ! indica un valor en simple precisión.
157.2	Representación alternativa del número anterior. A falta del símbolo #, el número está en simple precisión, aunque no esté el símbolo !.

Resumiendo, en Basic se utilizan constantes numéricas de estas clases:

Constante	Símbolo	Ejemplo
Entera	%	127%
Exponencial	E	127E2 = 12700
Doble precisión	D	127D2 = 12700
Doble precisión	#	12700#
Octal	&	&21
Hexadecimal	&H	&H21

Variables numéricas

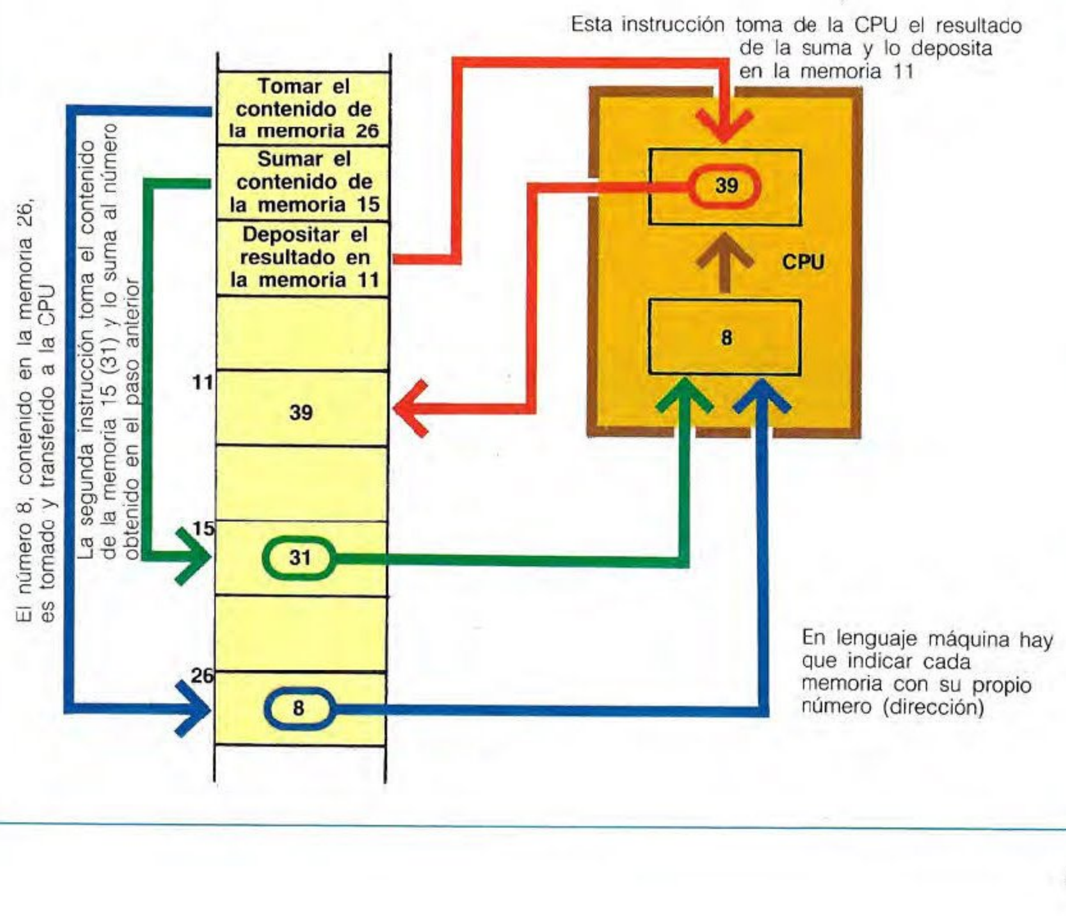
Las variables son magnitudes representadas con un nombre simbólico, cuyo valor cambia durante la ejecución del programa.

En las versiones de Basic más limitadas, los nombres de las variables son de dos caracteres como máximo; en las más amplias, de cuarenta. En la versión de dos caracteres, si se introduce un nombre más largo no hay señal de error: el sistema trunca automáticamente el nombre y sólo toma en consideración los dos primeros caracteres. Por lo tanto, hay que tener cuidado en no utilizar nombres de longitud superior a dos caracteres, pues, a causa del truncamiento au-

tomático, los nombres distintos que empiezan por las mismas dos letras, quedan iguales. Así, los nombres MES y MEDIA, utilizados para indicar dos variables distintas, en la versión de Basic más limitada se convierten en ME y, a lo largo del programa, las dos variables (MES y MEDIA) tomarían los mismos valores, con errores en el desarrollo de los cálculos.

El nombre de una variable es equivalente a una dirección de memoria; todas las instrucciones y valores con los que actúa han de estar contenidos en la memoria de la máquina, que puede considerarse dividida en dos zonas: en la primera residen los códigos correspondientes a las instrucciones; en la segunda, los valores. Para indicar qué valor se ha de usar hay que indicar su dirección, es decir, el número de la memoria que lo contiene. Por ejemplo, si se desea sumar los números 31 y 8, las instrucciones (ver gráfico inferior) han de hacer referencia a las direcciones de las memorias que contienen dichos valores (15 y 26 respectivamente); análo-

REFERENCIA A LA MEMORIA MEDIANTE DIRECCIONES ABSOLUTAS

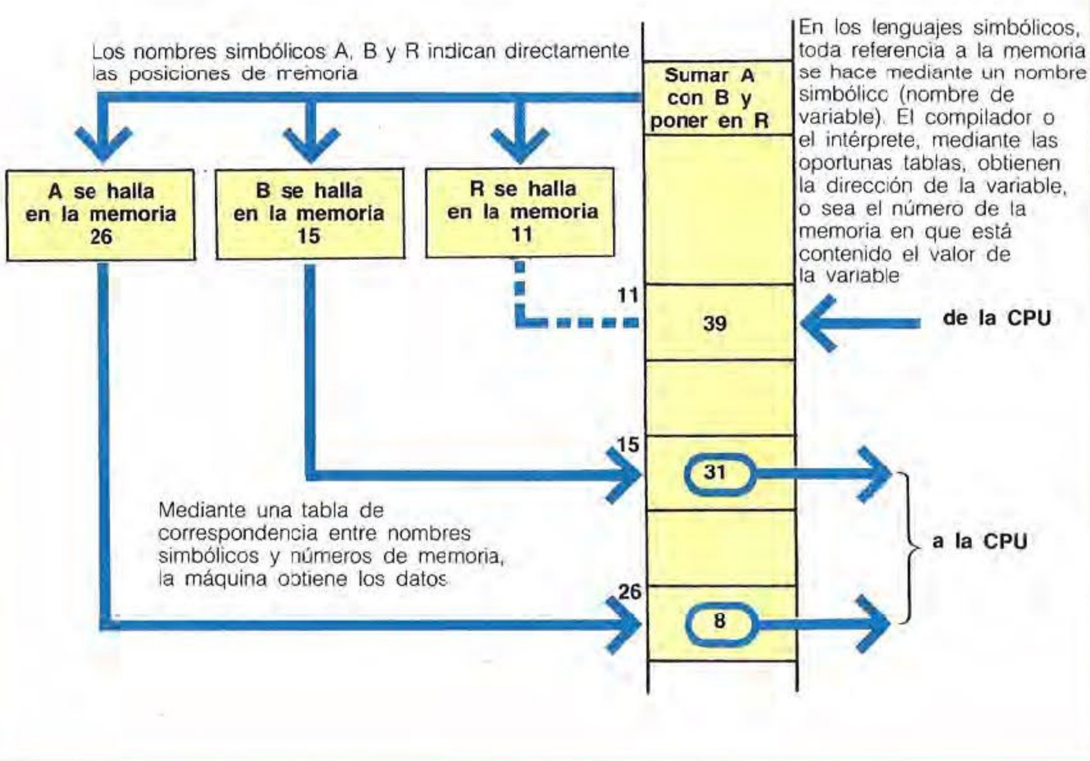


gamente, para indicar dónde ha de ser depositado el resultado, hay que indicar la dirección. En los lenguajes de alto nivel, las memorias se indican con nombres simbólicos (en el ejemplo, A, B y R), y el sistema crea una «tabla de correspondencia» en la cual, junto a cada nombre simbólico utilizado en el programa, escribe la correspondiente posición de memoria.

El programador no ha de preocuparse de conocer las posiciones ocupadas por los datos, puesto que puede llamarlos con su nombre simbólico: el intérprete Basic se encarga de encontrar su dirección. En Basic la operación del ejemplo (suma) se indica simplemente con R = A + B. El intérprete reconoce los símbolos = y +, y traduce la instrucción (R = A + B) en la oportuna serie de instrucciones en formato máquina, obteniendo las direcciones de las variables en la «tabla de correspondencia» (ver gráfico inferior). Las variables pueden ser:

- **enteras** (indicadas con el símbolo % junto a su nombre)

REFERENCIA A LA MEMORIA MEDIANTE NOMBRES SIMBOLICOS



- **en simple precisión** (símbolo !)
- **en doble precisión** (símbolo #)

La ocupación de memoria y la precisión (es decir, el número de cifras significativas) son las mismas que para las constantes numéricas.
El valor de una variable puede ser introducido con una instrucción de asignación (A = 21, B = 185, etc.) o como resultado de un cálculo (A = 3 * B + 2).
Al comienzo de un programa, todas las variables se igualan a cero y mantienen ese valor hasta el momento de usarlas. En el diagrama de flujo de la pág. 336, el bloque 500 tiene por objeto igualar a cero las variables Area total y Contador.
Esta puesta a cero se activa automáticamente al comenzar el programa, pero es conveniente ocuparse de ello de todos modos, en vista a futuras modificaciones del programa. Por ejemplo, si se quitara el bloque 500 y, posteriormente, se modificara el programa con un bucle para calcular dos áreas consecutivamente, el segundo cálculo sería erróneo por faltar la puesta a cero. La primera vez, al comienzo del programa, la puesta a cero es automática; la segunda vez (bucle para la segunda área) permanecen en las variables los valores anteriores, que, por tanto, se acumulan a los datos de la segunda área.

Ejemplos de variables

R! = 356.8	La variable R se define en simple precisión (símbolo !) y se le asigna el valor numérico 356.8.
A! = 37E20 * 8E20	Error. La variable A se define en simple precisión, mientras que el cálculo da como resultado un número que sólo puede ser contenido en una variable en doble precisión: 37E20 = 37 × 20^{20}, 8E20 = 8 × 10^{20}, y el producto 37 × 10^{20} × 8 × 10^{20} da 296 × 10^{40}, o sea 296E40; el exponente E40 no puede ser aceptado en simple precisión (el máximo es 38).
A # = 37E20 * 8E20	Notación correcta del cálculo anterior.
B! = 5/2E9	Error. El número 5 dividido por 2E9 (o sea, 2000000000) da un resultado demasiado pequeño para ser aceptado en simple precisión (siete cifras útiles).
C% = 721.3	Error. El símbolo % define la variable C como entera, por lo que no puede tener decimales.
C! = 721.3	Simbología correcta. También se puede usar la notación C = 721.3, puesto que la ausencia de símbolos significa, implícitamente, simple precisión.
R = 1240.21 #	Error. A la variable R en simple precisión no se le puede transferir un número definido en doble precisión (símbolo #), puesto que algunas cifras del número se perderían.

Conversiones entre los distintos tipos de variables

Durante el desarrollo de un programa puede ser necesario pasar una variable de un tipo a otro. En la conversión se producen redondeos o truncamientos de los valores numéricos. Los principales casos de conversión se dan:

- — en las instrucciones de asignación: A = 275
- — en los cálculos: A = 3.21 * 75.8 + B #
- — en las operaciones lógicas: (5.2 + 4) AND (3.7 + 2)

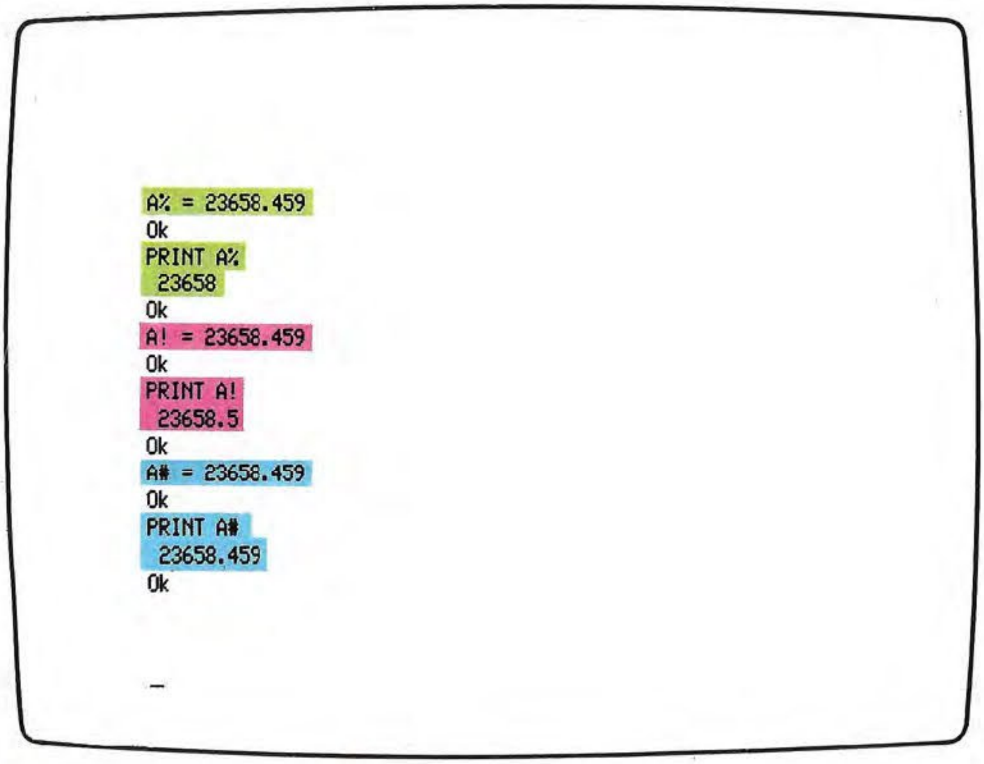
Redondeos

En los procedimientos de cálculo automático, la exactitud de los valores numéricos que se derivan de la ejecución de los cálculos puede depender en gran medida de los redondeos que la máquina efectúa automáticamente con los datos y con los resultados mismos.
Es, por tanto, muy importante saber a priori cuál es la magnitud del error que cabe esperar en el resultado de un determinado cálculo. La primera fuente de error es el redondeo efectuado por la máquina en la fase de asignación de las

PRECISION DE LAS VARIABLES

En pantalla se ven distintas asignaciones de tipo para una misma variable, A, cada una seguida por una fase de impresión.





■ La variable A es definida como entera.

■ La variable A es definida en simple precisión.

■ La variable A es definida en doble precisión. La fase de impresión subsiguiente evidencia un número de cifras significativas mayor que en el caso anterior, aún habiéndole asignado el mismo valor numérico.

constantes, y este error se puede ajustar utilizando la precisión más conveniente.

Redondeo en las asignaciones. El valor numérico se memoriza de acuerdo con el formato definido por la variable.
Por ejemplo:

A! = 152.6D3 Se transfiere a la variable el valor 152.6D3, luego se pierden las eventuales cifras que pasen de siete (por ejemplo, 1359.743298D4 se convierte en 1359.743E4).

A% = 28.3 A la variable entera A% sólo se transfiere la parte entera del número; el resultado es A% = 28.

A # = 562.5E4 La variable está en doble precisión (#), mientras que el número tiene simple precisión. En este caso las cifras no ocupadas por el número se igualan a cero.

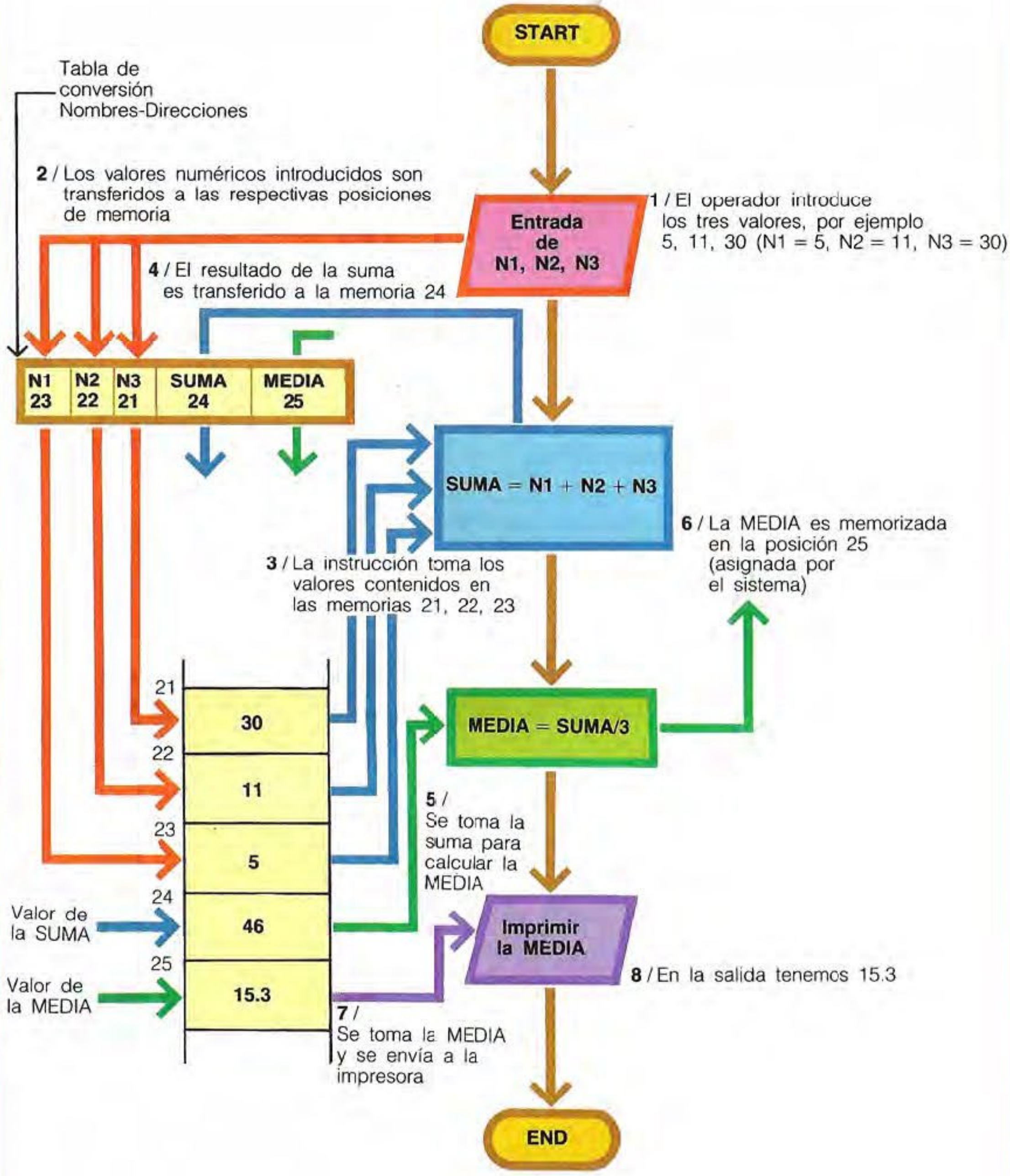
A% = 28.7 En este caso, al ser el decimal > 5, se redondea al valor superior. El resultado es A% = 29.

Redondeo en los cálculos. En las expresiones, todos los operandos se convierten a la misma precisión, igual a la más alta presente. Por ejemplo, la expresión A # = (7.56! + 8.26E4)/121D7 se evalúa como si todos los números estuvieran en doble precisión, puesto que el operando 121D7 lo está; también el resultado se memoriza en doble precisión (A # está en doble precisión).
Sin embargo, la expresión A! = (7.56! + + 8.26E4)/121D7 (igual a la anterior) se evalúa también en doble precisión pero se memoriza en simple precisión, que es la de la variable A!.

Resumiendo:

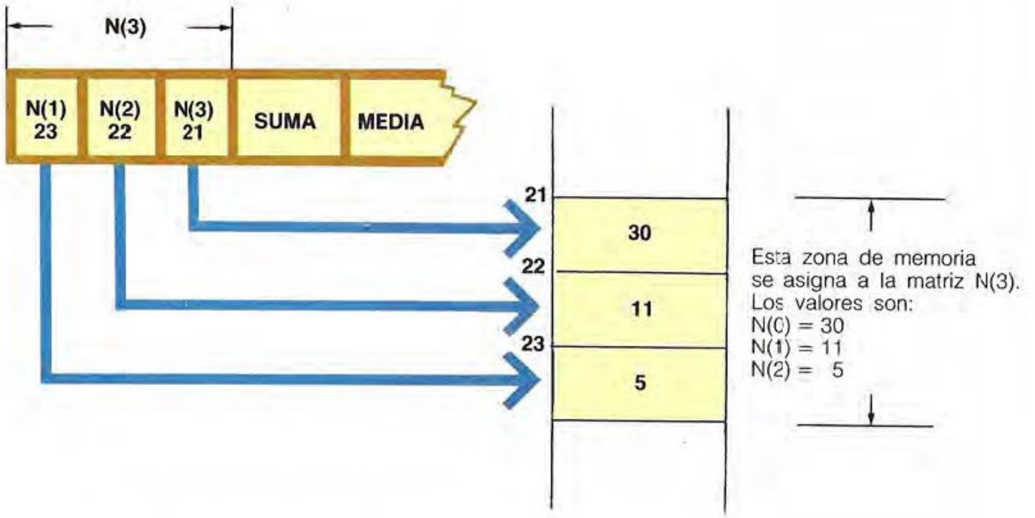
SI EL OPERADOR DESEA...	**HA DE INTRODUCIR...**
pasar a Basic	**MBASIC (o BA)**
definir una variable entera	**el nombre de la variable seguido por el símbolo % (ejemplos: A%, VALOR%)**
definir una variable en simple precisión	**el nombre de la variable seguido por el símbolo ! (ej., DIA!, C!)**
definir una variable en doble precisión	**el nombre de la variable seguido por el símbolo # (ej.: R #, MIO #)**
definir un número entero	**su valor sin decimales**
definir un número en simple precisión	**su valor seguido por el símbolo ! (ej.: 1275.3!, o bien, en notación exponencial, 12.753E2)**
definir un número en doble precisión	**su valor seguido por el símbolo # (ej.: 1275.3 #, o bien, en notación exponencial, 12.753D2)**
definir una función	**DEF FNX = ..., siendo X el nombre de la función (1 letra)**
determinar el resto de la división entre dos números	**Resto = Primer número MOD segundo número (ej.: 7 MOD 3)**
efectuar un cálculo inmediato	**PRINT cálculo (ej.: PRINT (7 + 3)/4)**
asignar un valor a una variable	**A = 7, DIA = 121.3**
pasar al sistema operativo	**SYSTEM**

DIRECCIONAMIENTO DE MEMORIAS CON VARIOS NOMBRES SIMBOLICOS



Mecanismos de traducción entre los nombres simbólicos de las variables (N1, N2, N3, SUMA, MEDIA) y las posiciones de memoria en que residen los valores numéricos de las variables

DIRECCIONAMIENTOS DE MEMORIAS CON UN SOLO NOMBRE SIMBOLICO



En vez de definir las tres variables N1, N2 y N3, se puede definir un solo nombre, N, que las incluya, es decir, que ocupe tres posiciones de memoria. La simbología es N(3). La matriz N tiene "dimensión 3", es decir, ocupa tres posiciones de memoria.

Redondeo en las operaciones lógicas. Los operandos que aparecen en una expresión lógica se convierten en enteros; si la conversión genera un número que excede el intervalo −32768/+32767, hay un error de desbordamiento («overflow»).

Por ejemplo, la expresión (5.2 + 4) AND (3.7 + 2) se evalúa como 9 AND 6 (3.7 + 2 = = 5.7, que se redondea como 6).

Variables estructuradas

En la programación se distingue también entre variables simples y estructuradas. En el segundo caso se trata de una implantación de software que permite hacer referencia a un conjunto de datos distintos utilizando el mismo nombre.

Los adecuados programas de sistema permiten reservar a una variable estructurada un determinado número de posiciones de memoria, en las cuales se memorizan de forma ordenada los diversos componentes de la variable.

Las variables estructuradas más sencillas previstas en el Basic son los **array** (las matrices «array») y las **cadenas**. Hay otros tipos de estructuración más complejos, de los que se hablará ampliamente más adelante.

Matrices «array». Con el término matriz o array se designa un grupo de memorias a las que se hace referencia con el mismo nombre simbólico. Supongamos que queremos escribir un programa que calcule la media aritmética de tres números introducidos con el teclado. Las funciones a realizar son:

- lectura de los tres números;
- cálculo de su media (suma de los números dividida por 3);
- impresión del resultado.

En principio, desarrollamos el diagrama de flujo llamando N1, N2 y N3 a los tres números. Los nombres simbólicos N1, N2 y N3 se asocian a tres memorias distintas (ver gráfico de la pág. 346). Al introducir (por parte del operador) los valores numéricos a asignar a las tres variables, el sistema toma la dirección de memoria a la que corresponden y deposita allí los valores introducidos con el teclado.

Las tres variables –N1, N2 y N3– pueden llamarse con el nombre único N(3), indicando con el símbolo (3) que a este nombre le corresponden tres posiciones de memoria (ver gráfico de pág. 347). La variable N(3) tiene, pues, varios valores simultáneos (tres, en este caso).
Para referirse a uno de estos valores hay que indicar el número progresivo.
La numeración interna de las matrices parte de 0, por lo que:

N(0) = Primer elemento de la matriz; corresponde a N1
N(1) = Segundo elemento; corresponde a N2
N(2) = Tercer elemento; corresponde a N3

Hay que tener cuidado de no confundir ambas simbologías. La notación N(3) se utiliza para informar al sistema de que hay tres valores (todos bajo el nombre N), mientras que los símbolos N(0), N(1) y N(2) se usan para tomar o depositar cada uno de los valores que constituyen la matriz N(3). A menudo resulta incómodo para el programador utilizar una numeración que empieza por 0: pensar en el primer elemento de una tabla como el elemento número 0 es contrario a nuestra lógica. En Basic hay una instrucción que modifica la base de numeración y la hace comenzar por 1. De esta forma, los elementos de la matriz anterior se convierten en N(1), N(2) y N(3).
Al hacer referencia a un elemento de una matriz se puede dar un índice al parámetro que lo identifica. Por ejemplo, el elemento N(2) (de valor 5) puede denominarse N(I), con I = 2. De esta manera, variando el índice I (I = 1, 2, 3) se pueden llamar todos los elementos de la matriz. Este método se usa mucho, puesto que permite «parametrizar» los programas.
En el cálculo de la media aritmética (ejemplo anterior), las líneas utilizadas son:

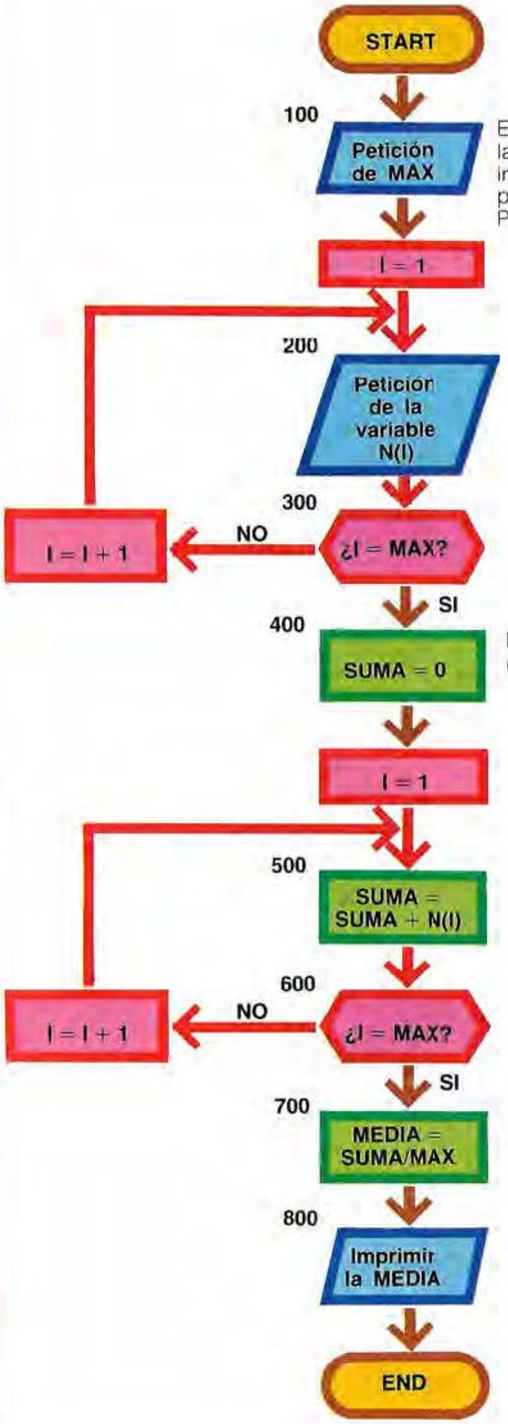
SUMA = N1 + N2 + N3
MEDIA = SUMA / 3
en el caso de variables separadas
SUMA = N(1) + N(2) + N(3)
MEDIA = SUMA / 3
en el caso de matriz (en base 1)

Si cambia el número de las variables (por ejemplo, si pasan a 8), estas líneas ya no son válidas y hay que reescribir el programa. Utilizando una matriz con índice no aparecen valores numéricos: el programa está parametrizado y se adapta, sin ninguna modificación, a distintas aplicaciones. En la página contigua se muestra el diagrama para el cálculo (parametrizado) de la media de un número cualquiera de valores.
La única limitación consiste en la amplitud de la memoria asignada a la matriz N. Por ejemplo, si la amplitud fuese N(3000), se podría calcular la media para un número de variables comprendido entre dos (para una sola variable la media no tiene sentido) y tres mil; el programa se adapta automáticamente.

Cadenas. La cadena es una serie de caracteres alfanuméricos (en código ASCII) entre comillas; por ejemplo, "NOMBRE, 1, 3, 5" es una cadena que contiene los caracteres: NOMBRE, 1, 3, 5. Los números que aparecen en las cadenas (salvo adecuadas transformaciones) no son utilizables para los cálculos, puesto que están representados en formato no numérico.
Las cadenas pueden ser constantes o variables, en el mismo sentido que cuando se trata de constantes y variables numéricas. Para informar al sistema de que un determinado nombre de variable se refiere a una cadena, se emplea el símbolo $ (dólar). Por ejemplo, NOMBRE$, DIA$, A$ son posibles nombres de cadenas. La longitud máxima de una cadena (variable o constante) es de 255 caracteres. La cadena que no contiene ningún carácter se denomina «cadena nula». Por ejemplo, A$ = "ADC" es una cadena (de nombre A$) que contiene los tres caracteres ADC; si escribimos A$ = " ", el símbolo A$ es el nombre de una cadena nula, ya que no contiene ningún carácter. Véase que A$ = " " no es una cadena nula, pues contiene un carácter: el carácter "espacio" (blank).
La longitud de las cadenas se adecua al número de caracteres que se introducen en ella. Definiendo en un punto del programa la cadena NOMBRE$ = "Pérez", se tiene una ocupación de cinco caracteres; si la misma cadena se asigna luego a otro nombre, poniendo, por ejemplo, NOMBRE$ = "Antonio García", su longitud pasa a catorce caracteres. Esta forma de funcionar se denomina «alojamiento dinámico», y es típica del Basic. El alojamiento dinámico del espacio reservado a las cadenas es útil porque exime al programador de la necesidad de asignar a priori una longitud a cada cadena. Pero su utilización puede causar problemas. Supongamos que escribimos un programa largo, cuya ocupación se aproxime a la totalidad

CALCULO DE LA MEDIA CON MATRIZ Y PARAMETRIZACION

Funciones I/O
Bucle
Cálculos



El programa pregunta con cuántos valores se va a calcular la MEDIA. En esta fase hay que comprobar que el valor introducido (MAX) sea inferior a las dimensiones previstas para la matriz [N(3000), luego MAX<= 3000]. Por ejemplo, supongamos que MAX = 128

En este bucle, el programa pide los valores de las variables. Comienza con I = 1, por lo que el valor introducido se atribuye a la primera variable N(I) = N1, con I = 1. Termina con I = MAX = 128. Por tanto hay que introducir **todos** los 128 valores declarados de las variables

Puesta a cero de SUMA (no es estrictamente necesario)

El bucle acumula (en la memoria denominada SUMA) la suma de los valores de las 128 variables. Inicialmente se pone SUMA = 0, por lo que a la primera vuelta (I = 1) se tiene SUMA = SUMA + N (1). En la segunda vuelta (I = 2) SUMA ya contiene N(1), por tanto:

SUMA = SUMA + N(2) = N(1) + N(2)

(SUMA contenía a N(1) de la primera vuelta)

Procediendo de esta forma se suman los 128 valores

El cálculo de la media es:

$$\text{MEDIA} = \frac{\text{Suma de los valores}}{\text{Número de valores}} = \frac{\text{SUMA}}{\text{MAX}}$$

Impresión del resultado

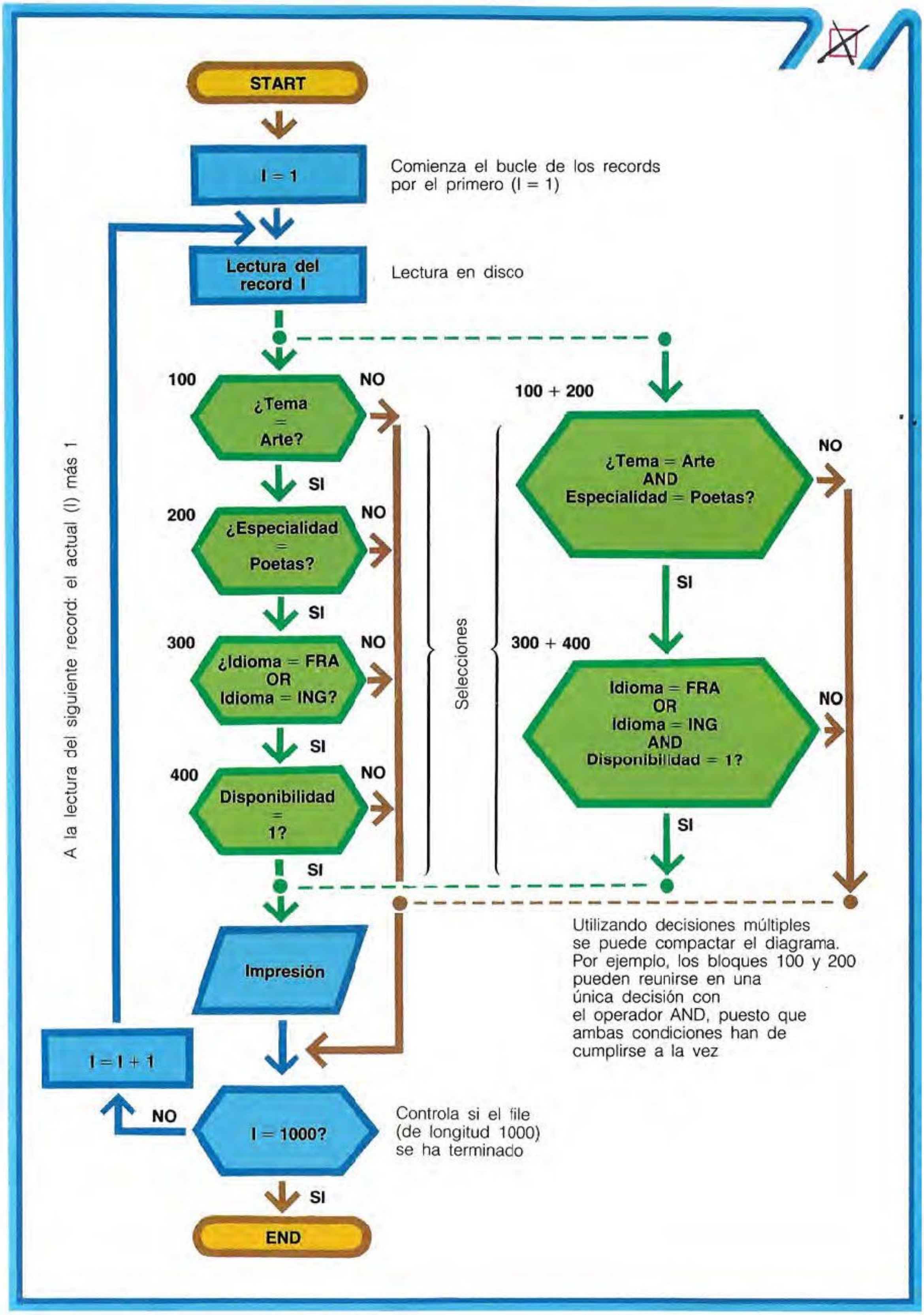
Soluciones del TEST 9

1 / La media de tres números es su suma dividida por 3. En la pantalla la instrucción es:
PRINT (5.2 + 15.21 + 7.43)/3
en impresora:
LPRINT (5.2 + 15.21 + 7.43)/3

2 / El operador AND tiene una prioridad inferior a la de los cálculos aritméticos; por tanto, en la expresión 3 + 2 AND 3 × 4 se efectúan primero los cálculos aritméticos (3 + 2 y 3 × 4). La expresión se convierte en 5 AND 12. Para hallar el resultado de esta expresión hay que pasar los números 5 y 12 a binario y luego aplicar el operador AND:
5 decimal = 101 binario
12 decimal = 1100 binario
5 AND 12 = 0100 = 4 decimal
Por lo tanto, 3 + 2 AND 3 × 4 = 4
En la segunda expresión, los paréntesis cambian el orden de prioridad; primero se efectúa la operación 2 AND 3, que da 2 (2 AND 3 = 10 AND 11 en binario). La expresión se convierte en: 5 + 2 × 4 = 13.

3 / Las funciones a efectuar son:
1 - lectura de un registro del file de datos;
2 - comprobar que el tema sea «arte»;
3 - comprobar que la especialidad sea «poetas»;
4 - selección de la lengua (FRA o ING = FRA OR ING);
5 - comprobar disponibilidad = 1 (presente);
6 - si las condiciones 2, 3, 4 y 5 se cumplen se puede imprimir el título de la obra;
7 - selección de un nuevo record (si el file no se ha acabado).
En la página contigua se muestra el diagrama de flujo del programa.

4 / La longitud de un record es la suma de los bytes ocupados por cada campo; en el ejemplo, 54 bytes (30 + 10 + 10 + 3 + 1 caracteres; recuérdese que 1 carácter = 1 byte). Aumentando a 70 la longitud del record (para futuras ampliaciones, por ejemplo, para la introducción del nombre del autor) el número de records (es decir, de volúmenes archivados) que puede memorizarse en un disco de 1 Mbyte (1000000 de bytes) es de 1000000/70 = 14285.
La capacidad de memorizar los datos de más de 14000 volúmenes no debe llevarnos a sobrevalorar las posibilidades del sistema, puesto que si creáramos realmente un archivo similar, los tiempos necesarios para su elaboración serían muy largos.
El acceso al disco requiere, como orden de magnitud, 0.1 segundos para cada record (hay que considerar los tiempos de posicionamiento de la cabeza lectora); por lo tanto, el tiempo de lectura de los 14000 records es 14000 × 0.1 segundos, o sea unos 25 minutos. El tiempo necesario para realizar las instrucciones es muy difícil de calcular. A título orientativo, se puede considerar que aumenta el tiempo total hasta 30 minutos.
La fase de impresión puede ser la más larga. Las impresoras de buena calidad poseen velocidades de 120 líneas por minuto; aun suponiendo que sólo haya que imprimir el 10% de los datos, tendríamos 1400 líneas (cada línea es un dato), que a la velocidad de 120 líneas por minuto requerirían 1400/120 = 12 minutos (si tuviéramos que imprimir el archivo entero, el tiempo necesario sería de 117 minutos, o sea casi 2 horas). Por lo tanto, el tiempo necesario para la ejecución del programa varía entre unos 45 minutos y unas 3 horas.


START
I = 1
Comienza el bucle de los records por el primero (I = 1)
Lectura del record I
Lectura en disco
100
¿Tema = Arte?
NO
SI
200
¿Especialidad = Poetas?
NO
SI
300
¿Idioma = FRA OR Idioma = ING?
NO
SI
400
Disponibilidad = 1?
NO
SI
Selecciones
100 + 200
¿Tema = Arte AND Especialidad = Poetas?
NO
SI
300 + 400
Idioma = FRA OR Idioma = ING AND Disponibilidad = 1?
NO
SI
Utilizando decisiones múltiples se puede compactar el diagrama. Por ejemplo, los bloques 100 y 200 pueden reunirse en una única decisión con el operador AND, puesto que ambas condiciones han de cumplirse a la vez
A la lectura del siguiente record: el actual (I) más 1
Impresión
I = I + 1
NO
I = 1000?
Controla si el file (de longitud 1000) se ha terminado
SI
END

de la memoria disponible, y que en el programa estén previstas como entradas un determinado número de cadenas (por ejemplo, los apellidos y las direcciones de una agenda). Siempre que las entradas no requieran un área de memoria excesiva, el programa funcionará normalmente. Si el operador introduce una serie de datos demasiado larga, se requiere para su alojamiento un área de memoria superior a la disponible, y el programa se bloquea. Para proteger los programas de este tipo de inconvenientes, se efectúa un control sobre el número de caracteres introducidos; un eventual exceso es señalado al operador (sin provocar el bloqueo del programa), que puede optar por una forma abreviada.

Las cadenas pueden dimensionarse como las variables numéricas. Así, si escribimos NOMBRE$(10) significa que reservamos un área de memoria para diez cadenas, cada una de las cuales puede tener desde un mínimo de cero caracteres (cadena nula) hasta un máximo de 255. Cada carácter ocupa un byte de memoria; por lo tanto, la ocupación global puede ir de 0 bytes (todas las cadenas nulas) a 10 × 255 = 2550 bytes (todas las cadenas llenas) y, a causa del alojamiento dinámico, no se puede saber a priori cuál será realmente la ocupación.

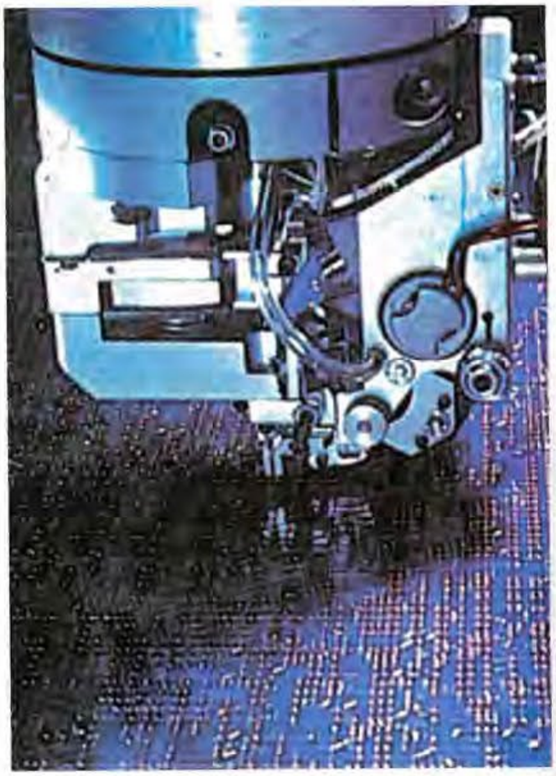
Cabe aplicar a las cadenas algunos operadores, que son:

- **el operador aritmético de suma (+),** con el significado de «encadenamiento» de dos cadenas. Por ejemplo, poniendo: A$ = "Antonio", B$ = "López", C$ = A$ + B$, el contenido de C$ es "Antonio López"; las dos cadenas (B$ y A$) han sido concatenadas, es decir, unidas una detrás de la otra, para formar una tercera cadena. En algunas máquinas, el símbolo de encadenamiento es &, por lo que la simbología pasa a ser C$ = A$ & B$, con el mismo significado anterior.
- **todos los operadores relacionales (=, < >, <, >, < =, > =).** Por ejemplo, si escribimos NOMBRE$ < "Pérez", la condición es cierta para todas las cadenas que contengan un valor menor que Pérez (López, Antonio, etc.), mientras que es falsa para las cadenas que contengan valores mayores (Vicente), según el orden alfabético.

Una aplicación de las cadenas se ve en los bloques 300 y 500 del diagrama de la pág. 331.

En el bloque 300 hay que verificar la igualdad entre el apellido leído en los datos y el valor "Pérez". El apellido es una variable (cadena), puesto que su valor cambia en el curso del programa, y puede ser indicada como APELLIDO$. Sin embargo, Pérez es una constante, y en la relación se escribirá entre comillas: "Pérez". La condición de igualdad se expresa escribiendo APELLIDO$ = "PÉREZ". También, en el bloque 500 la profesión es una variable, y podrá indicarse como PROF$, mientras que los valores Empleados y Docente son constantes. Por lo tanto, la relación se escribe así:

PROF$ = "EMPLEADO" OR PROF$ = "DOCENTE".



B. Coleman/Marka

Soldadora-robot que opera sobre un panel.

Subcadenas. Con este término se designa un grupo de caracteres tomados de una cadena. Si escribimos: A$ = "Antonio", B$ = "López", C$ = A$ + B$ da lugar a la cadena C$ partiendo de las cadenas A$ y B$. El procedimiento inverso, la extracción de los cinco segundos caracteres de la cadena C$, genera la subcadena "López", y la extracción de los siete primeros (Antonio) genera la subcadena A$.

En Basic hay algunas instrucciones que permiten la extracción de caracteres de una cadena, partiendo de una posición cualquiera y para cualquier número de caracteres.

Los comandos

En todos los lenguajes hay dos tipos de «órdenes»: los comandos y las instrucciones propiamente dichas. Los comandos se dedican a las funciones globales del sistema, como la puesta en marcha o la interrupción de un programa, mientras que las instrucciones propiamente dichas son la codificación de las funciones a realizar, o sea el programa mismo. En Basic, los comandos son más numerosos que en los demás lenguajes, y pueden utilizarse en su mayor parte incluso en el curso de un programa. En los demás lenguajes, el uso de los comandos en programación no es tan inmediato.
Los comandos Basic pueden dividirse en cuatro grupos de funciones homogéneas:

1 / Escritura y preservación de programas
2 / Impresiones y listas
3 / Activación e interrupción de programas
4 / Encadenamiento

Comandos para la escritura y preservación de programas

Cada instrucción Basic está numerada progresivamente, pero con paso arbitrario. Así, en un programa podemos usar una numeración que parta de 10 y aumente con incrementos de 5 (10, 15, 20, etc.); en otro podemos usar incrementos de 10 (10, 20, 30, etc.). Si escribiéramos los dos programas consecutivamente, sin desconectar la máquina, el resultado sería una mezcla de las instrucciones. Las de números iguales (pares) serían remplazadas por las nuevas; las otras (números impares: 15, 25, etc.), aunque pertenecientes al primer programa, serían englobadas en el segundo.
Para evitar recoger instrucciones no pertenecientes al programa en fase de escritura (eventuales residuos en memoria de programas anteriores) hay que introducir el comando NEW (nuevo). Esta orden informa a la máquina de que deseamos limpiar toda el área de memoria de usuario.
El comando NEW no puede ser impartido durante la digitación de un programa, puesto que causaría su destrucción.
Tras la introducción del comando NEW, toda el área de memoria de usuario queda limpia, y se pueden introducir las instrucciones que constituirán el nuevo programa.
En lugar de escribir personalmente los números de cada línea, se puede utilizar la numeración automática con el comando AUTO. El formato de la orden es:

AUTO línea inicial, paso

donde «línea inicial» es el número de la primera línea del programa, y «paso», el paso de numeración. Por ejemplo, el comando AUTO 120, 5 genera una numeración automática de las líneas que comienza en 120 y prosigue con paso 5. Durante la generación automática de los números de línea puede suceder que aparezca un número ya existente; en este caso el número aparece con un asterisco al lado, y el usuario puede interrumpir la numeración automática, salvando la línea preexistente, o puede continuar con la introducción, superponiendo la nueva línea a la anterior.
Las posibilidades de utilizar el comando NEW y de numerar a partir de un valor cualquiera son muy útiles en la escritura de programas largos. La escritura del programa puede interrumpirse y reemprenderse en cualquier momento iniciando la nueva numeración por un número de línea siguiente al último introducido.
El programa, una vez escrito, ha de ser preservado. Cualquier fallo de corriente en la red o, a veces, incluso una simple oscilación de tensión puede causar su pérdida (la memoria del ordenador es de tipo «volátil»: si falta la alimentación se destruye su contenido). Los medios utilizados para la memorización permanente son el diskette o la cinta magnética.
La estructura del comando para disco es:

SAVE "nombre del disco: nombre del programa"

El código SAVE es común para casi todas las máquinas e indica la función de «salvamento» de un programa. Los parámetros (nombre del disco y nombre del programa) dependen del sistema operativo. Por ejemplo, en PCOS, donde los dos discos se identifican con los números 0 y 1, el comando SAVE "O:PRUEBA" memoriza el programa que en ese momento reside en memoria en un file de nombre PRUEBA en el disco 0. El comando análogo en CP/M es SAVE "A:PRUEBA". El sistema operativo CP/M reconoce que el programa está en Basic y lo memoriza con la extensión BAS. Al pedir el directorio del disco (es decir, la lista de su contenido, con la orden DIR; ver Sistemas Operativos), el nombre completo de este programa será PRUEBA.BAS.
El comando SAVE, en la forma que acabamos de ver, no está completo. Hay dos maneras de

memorizar los programas: en formato ASCII y en formato binario compacto.
En formato ASCII, las letras y los números que constituyen las líneas del programa se memorizan como símbolos ASCII, es decir, como caracteres alfanuméricos sin codificación alguna; en el otro formato (binario compacto) cada carácter se codifica en binario.
En el momento de la introducción de las instrucciones por teclado, los caracteres son enviados en formato ASCII, pero una instrucción formada por caracteres ASCII no puede ser ejecutada, puesto que antes ha de ser codificada en binario. De esto se encarga el intérprete. Un programa puede ser memorizado en cualquiera de los dos formatos; será luego el intérprete quien decida si hace falta o no la codificación.
La simbología que define el formato de la memorización es la letra A para el ASCII y ninguna indicación para el binario. Así, el comando anterior puede adoptar las formas (CP/M):

SAVE "A:PRUEBA", A
memorización en ASCII
SAVE "A:PRUEBA"
memorización en binario

El formato binario puede utilizarse para proteger los programas. Introduciendo la letra P en lugar de A, el programa será memorizado en forma protegida, y podrá ser usado pero no listado ni corregido. La protección no puede revocarse, por lo que hay que conservar siempre una copia no protegida del programa. Por ejemplo, el comando SAVE "A:PRUEBA", P memoriza el programa PRUEBA de forma protegida; el programa puede ser usado, pero se vuelve inaccesible por lo que respecta a listados o correcciones. El formato binario ocupa menos espacio que el ASCII y, por tanto, es la forma más conveniente. Sin embargo, no puede usarse siempre, puesto que hay funciones, por ejemplo la compilación, que requieren el formato ASCII.
Veamos ahora con detalle la secuencia de comandos e instrucciones que hay que introducir a partir de la puesta en marcha del sistema para la escritura y memorización de un programa sencillo:

> **MBASIC**	A la puesta en marcha, la máquina se halla bajo sistema operativo. El símbolo > indica que está preparada. La introducción del comando MBASIC provoca la carga del intérprete Basic.
NEW	El usuario introduce el comando de cancelación de la memoria.
AUTO 50, 5	Petición de numeración automática a partir de la línea 50, con paso 5.
50 A = B + C **55 D = A * 6** **60 K = C – B**	La máquina presenta los números consecutivos y el usuario puede introducir las instrucciones del programa (A = B + C, etc.).
CTRL + C	Las dos teclas CTRL y C, pulsadas simultáneamente, indican el final de la fase de introducción (ver funciones de editado). El Basic puede aceptar nuevos comandos.
SAVE "A:PROG-1", A	El programa recién escrito se memoriza en el disco A con el nombre PROG-1 y en formato ASCII.
NEW	La memoria queda nuevamente borrada, y en ella no queda ni rastro de lo escrito anteriormente.

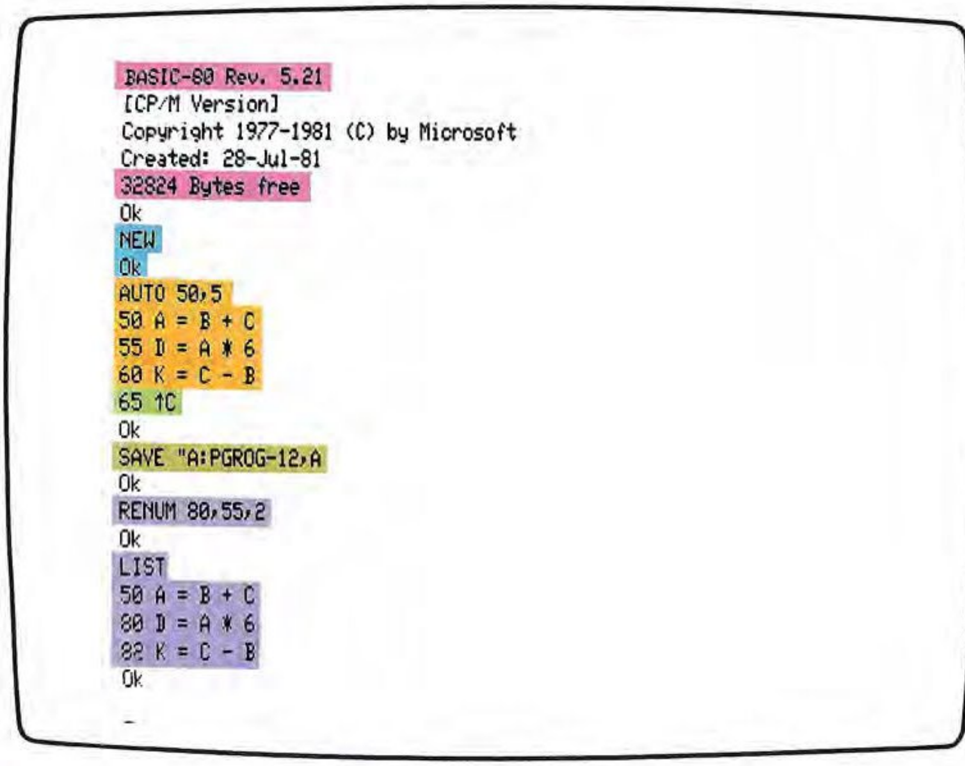
El programa PROG-1 reside ahora en disco, y para poder usarlo o corregirlo hay que volver a llamarlo a la memoria. La orden es LOAD "A:PROG-1". También para este comando, la forma es estándar, y las únicas diferencias tienen que ver con las distintas maneras de denominar las unidades de disco.
Tras la carga, el programa está de nuevo en la memoria y puede ser puesto en ejecución o corregido.
Los comandos para la corrección son RENUM Y EDIT.
Con RENUM se puede volver a numerar el programa entero o una parte del mismo.
El formato es:

RENUM n.° nuevo, n.° antiguo, paso

ESCRITURA Y MEMORIZACION DE UN PROGRAMA

En pantalla se ven las fases en que se articula el procedimiento completo.





Al comienzo, el sistema informa que se halla en ambiente Basic y que dispone de 32824 bytes libres.

El comando NEW (en este caso concreto no necesario) pone a cero el contenido de la memoria; el sistema responde entonces con el mensaje de comando ejecutado (OK).

A partir de este momento el operador puede digitar el programa.

La fase de introducción termina digitando CTRL + C. Esta orden se evidencia en pantalla en la forma ↑ C, y la línea correspondiente no es incluida en el programa (el intérprete se encarga de eliminarla).

La memorización en disco del programa (actualmente en memoria) se efectúa mediante el comando SAVE. En el ejemplo se ha cometido deliberadamente un error: al final del nombre PROG-1 que se quería dar al programa, se ha tecleado el carácter 2 en lugar de las comillas (") que indica final de cadena. Este error se puede cometer fácilmente, puesto que ambos caracteres están en la misma tecla (para digitar las comillas hay que pulsar simultáneamente la tecla SHIFT).
El sistema operativo no ha señalado error (es un defecto) y el programa se memoriza con un nombre distinto al deseado.

Estas líneas muestran la utilización de los comandos RENUM y LIST.

El «número nuevo» es el valor inicial de la nueva numeración, el «número antiguo» es la línea de programa en que comenzará la nueva numeración, y el «paso» es el de la nueva numeración. Por ejemplo, el comando RENUM 80, 55, 2 provoca la renumeración de las líneas a partir de la antigua línea 55 (que se convierte en la 80) con paso 2. Los nuevos números de las instrucciones pasan a ser 50, 80, 82 (la línea 50 queda invariada, puesto que la nueva numeración empieza en la 55).

El comando EDIT pone el Basic en la modalidad Editor, en la cual se pueden efectuar correcciones en las líneas del programa sin reescribirlas por completo.

La entrada en Editor no es la única manera de efectuar correcciones. La forma más simple consiste en reescribir la línea entera (sin entrar en Editor). Esto se logra introduciendo el número de la línea a sustituir seguido de la nueva instrucción; esta nueva introducción sustituye a la instrucción previa. Por ejemplo, introduciendo en el programa PROG-1 la línea 55 D = A/6, la instrucción 55 (D = A*6) queda sustituida por la nueva (D = A/6).

Si se hubiera utilizado el comando EDIT, se habría podido cambiar, simplemente, el antiguo símbolo * (multiplicación) por el nuevo / (división). Para correcciones tan nimias, el EDIT no ofrece ventajas considerables, pero en el caso de situaciones complejas o correcciones recurrentes puede ser muy útil.

Entrando en la modalidad EDIT, se dispone de cinco órdenes de edición:

1 / Desplazamiento del cursor. Permite correrlo a lo largo de la línea a corregir
2 / Inserción de caracteres
3 / Cancelación de líneas y de caracteres
4 / Búsqueda de caracteres
5 / Sustitución de caracteres

Las funciones que realizan estos comandos son muy similares a las del Editor del sistema operativo. El conocimiento profundo de los comandos EDIT sólo sirve en casos especiales; de todos modos, se describen las cinco funciones en la tabla siguiente, cuya lectura es, en cualquier caso, aconsejable a título informativo.

En el EDIT, además, se usan dos teclas especiales: RUBOUT y ESCAPE.

Principales funciones en EDIT (en la tabla de la pág. 358 se dan algunos ejemplos):

1 / Desplazamiento del cursor

El cursor (el indicador luminoso que señala la posición en que será introducido el carácter) puede desplazarse a lo largo de la línea a «editar» (es decir, a completar o corregir).

El desplazamiento hacia la derecha se obtiene con la barra de espaciado; hacia la izquierda, con la tecla RUBOUT.

2 / Inserción de caracteres en una línea

La inserción de uno o más caracteres en una línea se consigue posicionando el cursor sobre el último carácter anterior a la inserción y tecleando el código I seguido por los caracteres a insertar.

La función puede detenerse con la tecla RUBOUT. Si hay que insertar caracteres al final de una línea, y por tanto modificar su longitud, el código a usar es X. Introduciendo dicho código, el cursor se posiciona al final de la línea y la subsiguiente introducción del código I activa la inserción de los caracteres. La función de inserción al final de una línea termina pulsando la tecla ESCAPE.

3 / Cancelación de caracteres

Posicionando el cursor en el carácter a borrar, la función se activa en el código D. Por cada introducción de la letra D se borra un carácter. Este código puede complementarse con un factor de repetición; por ejemplo, escribiendo 4D se borran cuatro caracteres a la derecha del cursor.

El segundo código de cancelación es la letra H, que borra todos los caracteres a la derecha del cursor.

4 / Búsqueda de caracteres

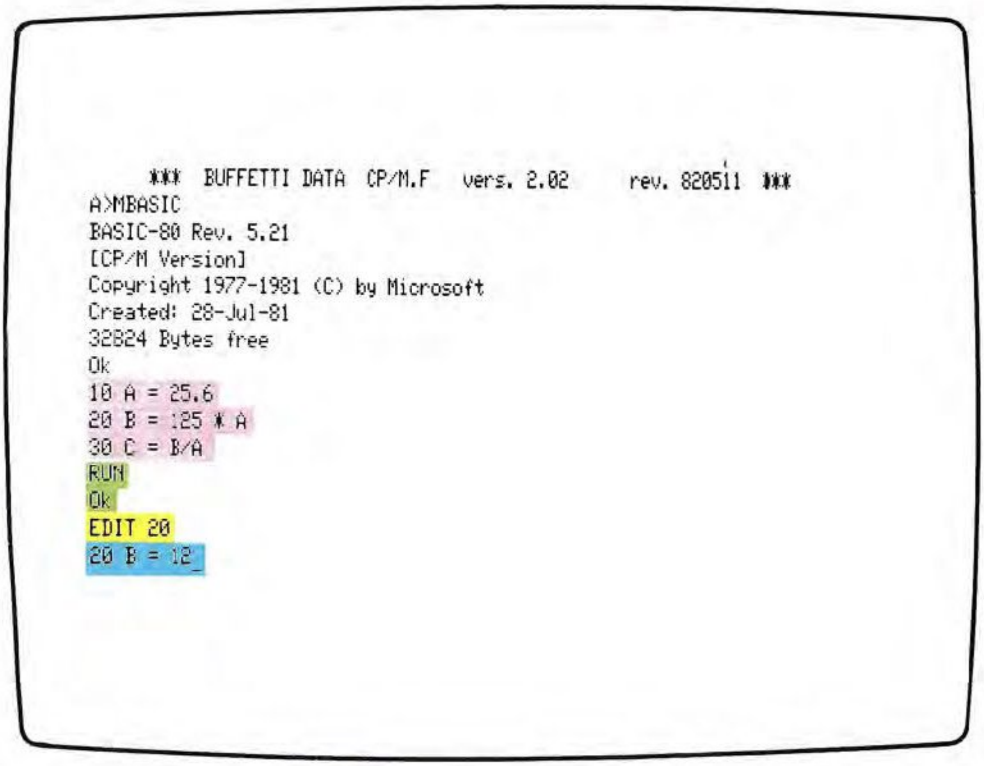
La función se activa en el código S. Por ejemplo, Sx busca el carácter x a lo largo de la línea examinada, a partir de la posición del cursor. Hay una segunda forma que, además de la búsqueda, lleva a cabo la cancelación (código K).

5 / Sustitución de caracteres

El código C sustituye el carácter indicado por el cursor por el nuevo valor suministrado junto con dicho código. Por ejemplo, Cr sustituye el carácter por la letra «r».

UTILIZACION DEL COMANDO EDIT





En estas líneas se introducen mediante el teclado las instrucciones que constituyen el programa.

El comando RUN pone en ejecución el programa residente en memoria. Al final de la ejecución aparece el mensaje de comando cumplido.

El comando EDIT 20 llama a la pantalla la línea 20 del programa residente en memoria.

Tras la aparición de la línea en pantalla, el operador posiciona el cursor bajo el carácter a modificar. Así, utilizando las opciones del Editor, se pueden introducir las correcciones deseadas. Al final, pulsando la tecla RETURN, la línea 20 así corregida se convierte en parte integrante del programa, y sustituye a la antigua línea 20.

PRINCIPALES FUNCIONES EN EDIT

1 / Desplazamiento del cursor
(1 desplazamiento por cada pulsación de la barra de espaciado)

10 V=A∗B−(C/D+6)

posición inicial del cursor → posición final del cursor

---

2 / Inserción de un carácter
Código = I

10 V=A∗B−(C/D+6)

Introduciendo el código I se inserta un carácter. Por ejemplo: I seguido por el carácter Z modifica la línea del modo siguiente:

10 V=A∗B−(C/DZ+6)

carácter insertado

---

3 / Cancelación de un carácter
Código = D

10 V=A∗B−(C/DZ+6)

El código D borra el carácter. La cancelación es evidenciada con un símbolo especial (\)

10 V=A∗B−(C/D\Z\+6)

Esta simbología indica que el carácter Z ha sido borrado. La línea 10 vuelve a ser como en los puntos 1 y 2

---

4 / Búsqueda de un carácter
Código = S

10 V=A∗B−(C/D+6)

El código S + busca el carácter +

El cursor se para bajo el carácter buscado

---

5 / Sustitución de un carácter
Código = C

10 V=A∗B−(C/D+6)

El código C sustituye el carácter + por el ∗

10 V=A∗B−(C/D∗6)

Mientras que la tecla ESCAPE (que normalmente se designa, en el teclado, con la sigla ESC) siempre existe, la RUBOUT puede faltar. Naturalmente, con un teclado en el que no exista esta tecla no se puede activar la función correspondiente.

Al utilizar el comando EDIT se ha de especificar el número de la línea a corregir; por ejemplo, el comando EDIT 55 activa las correcciones en la línea 55.

El último comando para la corrección de programas es: DELETE línea-línea, cuya función es borrar las líneas de programa comprendidas entre la primera y la segunda. Por ejemplo, la orden DELETE 10-28 borra todas las líneas 10 a 28, ambas inclusive.

En algunas máquinas, para este comando, el símbolo de separación entre los dos números de líneas es el guión; en los demás comandos es siempre la coma (por ejemplo, RENUM 20, 28, 5). Esta diferencia reduce las posibilidades de error en el uso de la función DELETE.

Wafer, chip & Co.

El microprocesador es probablemente el producto más extraordinario de la revolución del silicio: en una plaqueta, o chip, de silicio, condensa una capacidad de cálculo para la cual, en 1950, habría hecho falta una maquinaria de cuarenta toneladas. Además, puede reproducirse en millones de ejemplares, con un costo unitario de unos pocos dólares.
Añádanse al microprocesador algunos microcircuitos más para la memorización, un sistema que suministre la energía necesaria y algún otro accesorio funcional, como un teclado y una unidad de visualización, y se tendrá un ordenador. Pero ¿qué es un chip? ¿Qué puede hacer y cómo está hecho?
Un chip es un fino rectángulo de silicio sobre cuya superficie, apenas visible con microscopio, aparece el intrincado diseño de un microcircuito que se extiende por la masa del chip en una serie de estratos de silicio puro o impuro (dopado), óxido de silicio y aluminio.
El elemento básico del chip es el transistor, concretamente el transistor de efecto de campo (FET: Field Effect Transistor). *Un transistor con efecto de campo tiene tres conexiones eléctricas, denominadas «fuente»* (source), *«drenador»* (drain) *y «puerta»* (gate). *La corriente puede pasar de la fuente al drenador sólo cuando se aplica la tensión adecuada al electrodo puerta. Así, el transistor de efecto de campo funciona de forma similar a un interruptor de botón en el que la tensión aplicada a la puerta equivale al dedo que aprieta el botón. Conectando juntos un determinado número de estos transistores de efecto de campo, se puede lograr el desarrollo de las operaciones lógicas simples que constituyen la base del cálculo automático.*
Un transistor de efecto de campo está formado por dos clases distintas de silicio, la N *y la* P. *El silicio* N *está «dopado» con una impureza (como el fósforo) con objeto de producir en él un exceso de electrones, mientras que el silicio* P *tiene pocos electrones, lo que se consigue dopándolo con otra impureza (como el boro). Dos zonas de silicio* P *en contacto con una zona de silicio* N *constituyen un típico transistor de efecto de campo. Las dos zonas del tipo* P *forman la fuente y el drenador, mientras que un electrodo aplicado a la zona del tipo* N *entre la fuente y el drenador constituye la puerta. Cuando se fabrica un microcircuito, que puede contener miles de transistores de efecto de campo, el soporte básico u oblea (denominado* wafer*) se hace con uno de los dos tipos de material, por ejemplo el* P, *y con el «dopado» se crean islas del otro tipo allí donde es necesario. Las zonas que no han de doparse son oportunamente protegidas.*
La producción de un nuevo microcircuito comienza con la especificación del producto y con un equipo de proyecto. El equipo incluye especialistas en electrónica, que han de diseñar los circuitos de forma que realicen las funciones requeridas, y expertos en producción.
Tras haber sido diseñados, los circuitos son sometidos a pruebas mediante simuladores especiales, para ver si dan los resultados deseados. Una vez que la simulación ha dado resultados satisfactorios, los circuitos son trazados con ayuda de un ordenador gráfico, y las distintas partes de los elementos del circuito son asignadas a determinados estratos del paquete de silicio. El ordenador produce también un sistema de control a gran escala, además de una cinta magnética codificada para controlar los aparatos que producen las máscaras.
Las máscaras se utilizan para trazar el circuito en los distintos estratos del paquete. Antes, las máscaras se obtenían con procedimientos fotográficos, reduciendo progresivamente las dimensiones de un esquema muy aumentado del circuito. Actualmente, una máquina de haz de electrones, controlada por la cinta magnética codificada, traza el esquema del circuito directamente sobre la máscara.
Como ya hemos dicho, los microcircuitos se realizan sobre una oblea de silicio. Para preparar las obleas se funde en un crisol silicio puro, al que se añade la cantidad necesaria de «impureza». El silicio fundido se extrae lentamente y se convierte en un único gran cristal cilíndrico; en un lado se realiza un plano alineado con la estructura cristalina, para permitir una grabación precisa durante el tratamiento. Luego se corta el cristal de forma que se obtengan elementos circulares (obleas) de unos 100 o 120 mm de diámetro. Tras ser pulimentada con gran precisión, la superficie de una oblea está lista para acoger los estratos del circuito. Cada oblea puede contener hasta 500 microcircuitos. Un estrato típico del microcircuito podría obtenerse de la forma siguiente: en primer lugar, se crea una capa de bióxido de silicio enviando vapor de agua o un gas rico en oxígeno sobre la superficie de la oblea, en un horno. Regulando

La fabricación de un microcircuito de silicio comienza con una oblea de silicio puro. En este dibujo vemos una sección imaginaria.

La superficie de la oblea se transforma en óxido de silicio mediante un tratamiento en horno con vapor de agua o con un gas rico en oxígeno.

La capa de óxido de silicio se reviste con un producto químico fotosensible que se endurece al exponerlo a los rayos ultravioleta.



Sobre la superficie se pone una máscara con el modelo de los circuitos, y los rayos ultravioleta endurecen las zonas fotosensibles.

La máscara se quita y las zonas no endurecidas del material fotosensible se eliminan mediante tratamiento con un revelador químico.

Las zonas de óxido dejadas al descubierto por el material fotosensible son atacadas por el ácido y eliminadas revelando la superficie del silicio.



Los residuos de la sustancia fotosensible se eliminan con el revelador, dejando intacto el óxido de silicio.

La oblea se calienta en un horno que contiene un elemento químico que se difunde sobre las zonas de silicio descubiertas, creando las zonas semiconductoras.

Tras muchos tratamientos análogos al descrito, se deposita una capa de aluminio, también sometida a máscaras y a la acción del ácido para formar las conexiones.

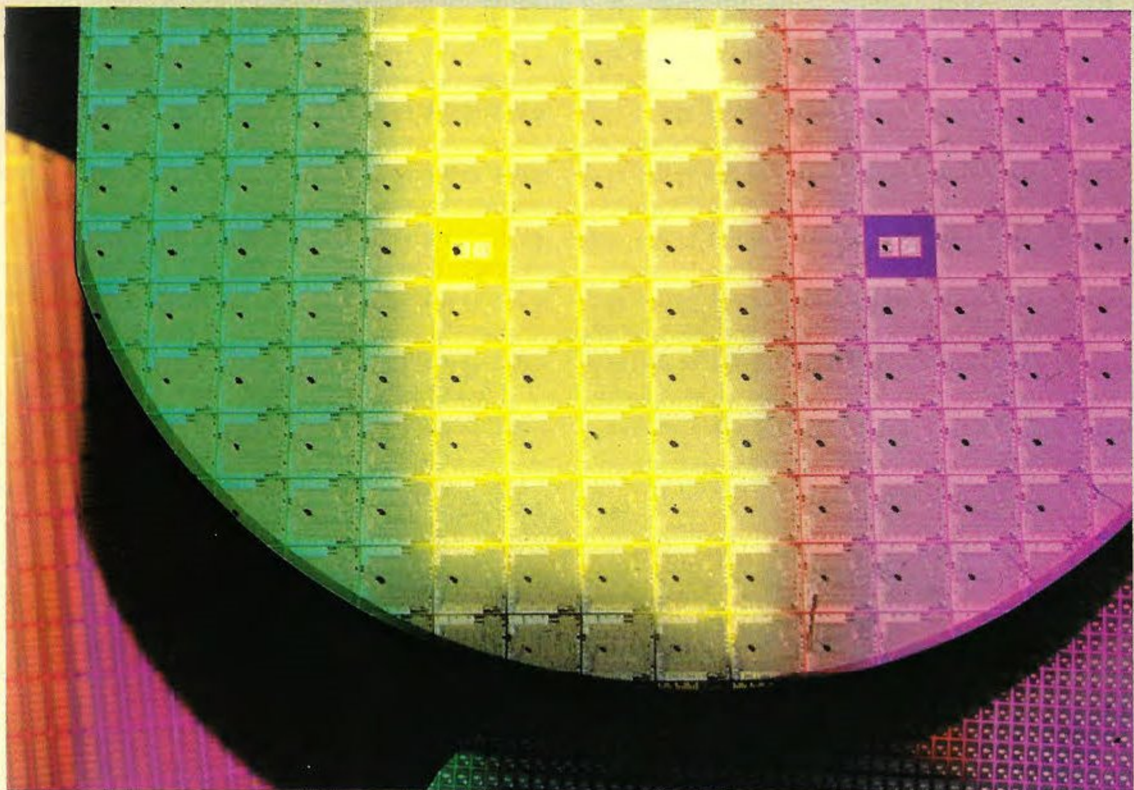


Advertising Arts

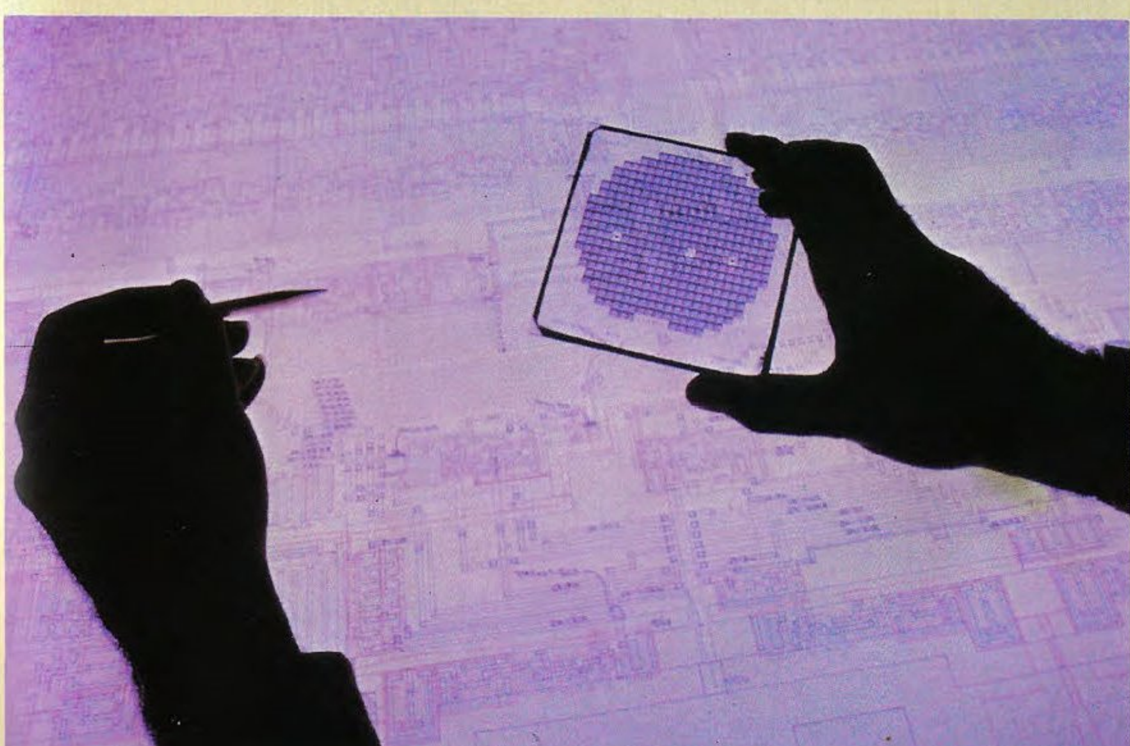
la temperatura y la cantidad de gas, se consigue una capa de óxido del espesor deseado. La capa de óxido se reviste luego de un producto químico fotoendurecible, es decir, que se endurece al exponerlo a los rayos ultravioleta.
La radiación ultravioleta se envía sobre la oblea a través de una máscara adecuada; para todas las obleas se utiliza la misma máscara, por lo que ésta lleva el mismo diseño repetido centenares de veces. El modelo de la máscara se aplica mediante impresión por contacto o por proyección. En el primer caso, la máscara se apoya directamente sobre la oblea; en el segundo, un sistema óptico proyecta el diseño sobre ella. Cualquiera que sea el sistema utilizado, el material se endurece allí donde incide la luz. El material de las zonas excluidas del endurecimiento es disuelto y eliminado mediante un revelador químico. La operación siguiente es la grabación con ácido. El óxido de las zonas de la

oblea expuestas se disuelve; las zonas revestidas de película fotorresistente permanecen inalteradas. El agente empleado es el ácido fluorhídrico. Finalmente, el residuo del material fotoendurecedor se elimina con el revelador. Este último deja una serie de zonas de óxido que cubren la oblea donde no es necesario el dopado. Para aplicar el dopante, la oblea se calienta en un horno en atmósfera de fósforo o boro, según deban formarse regiones N o regio-

Microcircuitos impresos en una oblea de silicio, antes de ser separados. Abajo: En cada chip de la oblea se imprime el circuito que se ve en el dibujo del fondo.



B. Coleman/Marka



B. Coleman/Marka

nes P. La oblea permanece en el horno hasta que un número suficiente de átomos de dopante ha penetrado en las zonas de silicio expuestas. Las operaciones descritas producen un estrato del circuito, y se repiten, con algunas variaciones, para producir todos los demás estratos (un microcircuito típico incluye once estratos, que requieren unas sesenta operaciones como las descritas). Finalmente se deposita una capa de metal (por ejemplo, aluminio) que, tras haber sido sometida a un proceso de ataque con pantalla, forma una red de conexiones entre las diversas zonas de tipo N y P, completando así el circuito. El estrato metálico incluye también un cierto número de contactos que permiten conectar el microcircuito con el exterior. Por regla general, el microcircuito se protege con un estrato final de óxido.

A causa de la extrema complejidad y reducidísimas dimensiones de los chips, la más mínima contaminación en un estadio cualquiera del proceso de preparación podría malograr el circuito. Por este motivo todas las operaciones se realizan en ambientes limpísimos cuyas condiciones son controladas estrictamente mediante filtraciones del aire y regulación de la temperatura. También los técnicos llevan indumentarias protectoras poco menos que esterilizadas.

Tras depositar la capa protectora de óxido se procede a la comprobación de las plaquetas de la oblea. Se conecta sucesivamente a los contactos de los diversos microcircuitos una serie de terminales, mientras un ordenador realiza el programa de pruebas. Los chips que resultan defectuosos son marcados automáticamente con tinta y eliminados en la fase de separación.

Al principio, la separación se efectuaba partiendo la oblea a lo largo de la línea de corte del cristal, pero las técnicas modernas se valen de sierras de diamantes o del láser. Los microcircuitos que superan la prueba son encolados sobre un bastidor; después, se sueldan finos hilos de oro a los terminales de los microcircuitos y a los previamente situados en el soporte. El conjunto se protege con una envoltura de plástico o cerámica y se somete a nuevo control.

Puede suceder que los chips de una nueva producción estén libres de defectos sólo en un 5%, por lo que la mayor parte de los microcircuitos resulta inutilizable y ha de ser descartada. Incluso una buena producción puede tener una tasa de validez no superior al 25%. Se han conseguido mejoras rediseñando las zonas defectuosas de los circuitos, modificando los procesos y, sobre todo, reduciendo las dimensiones de los chips. Aparte del evidente aumento del número de microcircuitos producidos con el mismo proceso, se obtiene también un aumento del rendimiento, puesto que, a igual número de causas de defecto (gránulos de polvo, imperfecciones superficiales, etc.) resultará dañado un porcentaje menor de chips. Disminuyendo la longitud de las conexiones, aumenta la velocidad de funcionamiento del dispositivo, factor a tener en cuenta en los circuitos de los ordenadores, que efectúan millares de cálculos por segundo.

Antes de hacer previsiones para el futuro, vale la pena mirar hacia atrás, a mediados de los años sesenta, cuando los microcircuitos estaban en el comienzo de su «carrera». Gordon Moore, fundador de Intel, predijo que el número de componentes incluibles en un solo microcircuito se duplicaría cada año. Pocos le creyeron, pero hasta el momento, la «ley de Moore» ha resultado bastante exacta. En 1965 el número de componentes de un solo microcircuito era de unos 32; en 1980, el número en laboratorio se aproximaba a un millón.

Es probable que los microcircuitos continúen cumpliendo la ley de Moore. Esto será más fácil para los elementos de memoria, puesto que siguen un esquema más regular. A medida que en un microcircuito se incluya un número mayor de elementos, las técnicas de producción tendrán que cambiar. Por ejemplo, pronto será imposible utilizar rayos ultravioleta en las fases que requieren máscaras: la longitud de onda es demasiado grande para los circuitos más intrincados. Sin embargo, seguramente el problema podrá resolverse recurriendo a los rayos X, cuya longitud de onda es mucho menor.

Mirando aún más lejos, los científicos están examinando las posibles alternativas al silicio como material básico para la realización de los chips: el arseniuro de galio es uno de los candidatos; se considera que permitirá realizar microcircuitos mucho más veloces con menos requisitos de energía. La reducción del consumo es importante, porque significa que los elementos del circuito no dispersarán tanto calor y podrán, por lo tanto, ser todavía más pequeños. Y cuanto menores sean los elementos, tanto mayor será el número de los mismos que se podrá concentrar en una plaqueta. Sin embargo, los circuitos de arseniuro de galio no son tan fáciles de manipular como los de silicio.

Comandos para la cinta magnética. El soporte de memorización alternativo al disco es la cassette magnética, y también para este tipo de memoria masiva sirven las órdenes de memorización (SAVE) y de carga (LOAD).
En el caso de la cassette las órdenes tienen la forma:

CSAVE "NOMBRE"
CLOAD "NOMBRE"

donde NOMBRE es el nombre del programa. En este caso, sin embargo, sólo el primer carácter (en el ejemplo, N) se utiliza para dar nombre al file; por tanto, los comandos SAVE "NOMBRE" y CSAVE "N" son equivalentes. Para ambos, el programa se memoriza con el nombre N.
Hay que tener cuidado de no usar nombres con la misma inicial para programas distintos, puesto que si se usaran simultáneamente, serían memorizados con el mismo nombre y el segundo recubriría al primero.
Las instrucciones CSAVE y CLOAD se utilizan de forma ligeramente distinta también para memorizar datos, en forma de cadenas de caracteres, también en cassette.

Impresión y listado de programas

En el Basic estándar se reconocen sólo dos dispositivos de salida: la pantalla y la impresora. Todas las operaciones de listado de programas o de impresión de los datos pueden direccionarse a uno u otro periférico.
Para los programas, los comandos son:

LIST línea inicial, línea final
(para la pantalla)
LLIST línea inicial, línea final
(para la impresora)

«línea inicial» y «línea final» son los números de la primera y última línea del bloque de líneas a imprimir.
Por ejemplo, la orden LIST 20,100 genera en pantalla la lista del programa residente en memoria en ese momento, desde la línea 20 hasta la 100. La orden equivalente para la impresora es LLIST 20,100.
Hay, por último dos comandos poco usados: NULL y WIDTH. NULL sirve para especificar cuántos espacios en blanco hay que dejar entre una línea y otra. Por ejemplo, NULL 4 significa que al final de cada línea se dejan cuatro espacios en blanco. El uso de esta orden está restringido a los sistemas que disponen de perforadora de cinta.
La orden WIDTH impone una longitud máxima en la línea en fase de impresión. Por ejemplo, al introducir WIDTH 20, cualquiera que sea la longitud de la línea a imprimir, sólo son escritos los primeros 20 caracteres. La orden puede dirigirse tanto a la pantalla (en la forma que acabamos de ver) como a la impresora, completándola con la opción LPRINT. El comando WIDTH LPRINT 20 crea una ventana de 20 caracteres en la impresora; los eventuales caracteres que excedan de los 20 habilitados son ignorados.

Activación e interrupción de programas

Un programa residente en memoria se pone en ejecución mediante el comando RUN.
Todos los programas contienen la instrucción de terminación (END) y normalmente se detienen, tras haber realizado su tarea, con esta instrucción. Puede suceder, especialmente en las pruebas iniciales, que algún error de programación cree caminos distintos de los previstos que no pasen por la instrucción END.
La máquina no encuentra entonces la instrucción de terminación, y el programa sigue girando indefinidamente. Para detenerlo hay que introducir simultáneamente los códigos CNT (o CTRL) y C (Control + C). La ejecución puede ser reemprendida con la orden CONT (continuar). El comando CONT también es útil en la fase de control del funcionamiento de los programas. En los puntos a controlar se inserta una instrucción STOP que detiene el programa. El operador puede entonces analizar los resultados intermedios pidiendo su impresión de forma inmediata. Para proseguir la ejecución desde el punto en que ha sido interrumpida, hay que introducir el comando CONT; de esta manera se pueden comprobar todos los pasos intermedios de cada función y controlar su validez.
El comando CONT no puede ser utilizado si en el momento en que el programa se detiene se efectúan correcciones.
Las últimas dos órdenes de este grupo, muy usadas también en los programas como instrucciones, son TRON y TROFF.
El comando TRON activa el proceso de visualización de los números de las instrucciones a medida que son efectuadas. De este modo se tiene una visión del camino seguido por la máquina en el interior del programa.

Un programa puede contener muchas «opciones» y «decisiones», en función de las cuales se activa un recorrido en lugar de otro. Las lógicas pueden llegar a ser tan complejas que no permitan, en caso de error, una interpretación fácil. En muchos casos conviene seguir, con el comando TRON, el camino que la máquina está recorriendo realmente antes de evaluar todas las posibilidades analizando los diagramas de flujo o el programa.
El comando TROFF desactiva la visualización de la «trayectoria del programa».

Encadenamiento

Utilizando la técnica de fraccionamiento de los programas en subrutinas, es normal que algunas partes, escritas para una determinada aplicación, puedan ser adaptadas también a otros casos.
En previsión de un uso posterior, conviene siempre memorizar las rutinas de utilización general en files separados, como si fueran programas en sí mismas. Cuando hacen falta, en vez de reescribirlas se toman del disco y se unen al programa residente en memoria. La instrucción es MERGE «nombre del file». La instrucción se refiere a files que contienen programas y, por tanto, en Basic. Si el nombre corresponde a un file de datos, el comando no es ejecutado y se señaliza error.
La instrucción MERGE "PRUEBA" carga, tomándolo del disco, el programa memorizado en el file PRUEBA y lo une al programa que está en la memoria en ese momento.
El file ha de estar en formato ASCII; por tanto, si había sido memorizado en binario, se genera un error y la operación MERGE no se efectúa. Durante el MERGE, si hay números de línea iguales entre el programa en memoria y el que se toma del disco, se produce la sustitución de la instrucción en memoria por la tomada del disco. Por tanto, antes de efectuar un MERGE, hay que cerciorarse de que no haya números de línea iguales; si hay una superposición, se puede volver a numerar el programa en memoria.

Ejemplo:

Programa en memoria
```
10 A = B + C
20 D = A * 2 + SQR (B)
30 E = A + D
40 TOTAL = E * 2
50 T1 = E - TOTAL
```

Programa en disco con el nombre PROG-2:
```
30 F = A + B +C
50 SUMA = TOTAL + F
60 END
```

La instrucción MERGE "PROG-2" carga PROG-2 desde el disco y lo une con el programa residente en memoria. Las anteriores instrucciones 30 y 50 son sustituidas por las contenidas en PROG-2 y se añade la línea 60.
Un programa bien estructurado consta de una línea principal (main) y de una serie de subrutinas que realizan las distintas funciones. El main ha de contener sólo las partes generales y las llamadas a las subrutinas. Una estructura de ese tipo se presta al uso del MERGE. En la fase de escritura de las diversas partes, en lugar de escribir y memorizar todo el programa en un único file, se preparan las partes (main y subrutinas) por separado, e igualmente se memorizan por separado.
El programa entero puede construirse cargando en memoria el main y conectando todas las rutinas necesarias con sucesivos MERGE. Las mismas rutinas podrán utilizarse para otras aplicaciones procediendo de forma análoga, sin más que sustituir el main.
Otro comando es CLEAR. Realiza las siguientes funciones:

- Cierra los files (los files cerrados no pueden ser utilizados por los programas; para acceder a ellos nuevamente hay que reabrirlos)
- Pone a cero las variables
- Anula el contenido de las cadenas
- Fija un límite máximo a la memoria disponible para el usuario
- Define el área máxima disponible para el stack

Este comando se usa poco, ya que las puestas a cero y las subdivisiones de las áreas de memoria se efectúan automáticamente al cargar y poner en ejecución un programa.

Las instrucciones

Las instrucciones se obtienen codificando en uno de los posibles lenguajes de programación las acciones que el ordenador ha de realizar. En este capítulo se presentan todas las instrucciones genéricas que operan con variables o constantes. Las instrucciones que actúan sobre las

TEST 10

1 / Un programa ha sido memorizado en disco de tres formas distintas, utilizando los siguientes comandos:

a) SAVE "A:PRUEBA"
b) SAVE "A:TEST", A
c) SAVE "B:PRUEBA2", P

¿Qué versión está en ASCII?
¿En qué disco se memoriza la versión de nombre PRUEBA2?
¿En cuáles de las tres versiones se pueden efectuar correcciones?

2 / El siguiente programa:

```
10 A = 6
20 B = 8
30 C = 10
40 D = C * (A + B)
50 E = 2 * D/A
```

se vuelve a numerar con el comando REN 20,10,2 y luego se une (con el comando MERGE) al programa:

```
20 B = 2
20 F = D + E
```

Decir qué valores toman las variables D y E en los casos siguientes:

a) haciendo correr el primer programa sólo antes de la nueva numeración.
b) haciendo correr todo el programa tras la nueva numeración y unión de la primera parte con la segunda.

3 / ¿Cuál es el programa que queda escrito al final de los siguientes comandos?

```
NEW
10 A = 3
20 B = 2
30 C = 3 * (A + B)
REN 10,10,2
AUTO 20,5
20 F = 2 * (C + A)
25 PRINT F
```

4 / ¿De qué manera la línea 20 F = 2 * (C + A) puede ser sustituida por la F = 4 * (C + A)?

5 / ¿Qué aparecerá en la pantalla al poner en ejecución el siguiente programa?

```
10 A = 6
20 TRON
30 B = A + 2
40 C = 2 * B
50 TROFF
60 D = B + C
```

¿Cuál es el comando para memorizar este programa en el disco A en formato ASCII, con el nombre PRUEBA?

Las soluciones en las págs. 374 y 375.

Cuadro sinóptico de los comandos

Comando	Objeto	Ejemplo
NEW	Pone a cero el contenido de la memoria	NEW
AUTO n, m	Plantea la duración automática de las líneas introducidas a partir de la línea número n y con paso m	AUTO 10, 5
CLEAR	Pone a cero variables y cadenas	CLEAR
SAVE "X:YYY", Z	Memoriza en el disco X (A/B o 0/1) el programa de nombre YYY con opción Z (Z = A → ASCII; Z = P → protegido); Z no indicado → binario)	SAVE "A:PROG-1", A SAVE "A:PROG-2" SAVE "0:NOMBRE", P
RENUM n1, n2, m	Vuelve a numerar a partir de la línea n2, asignándole el nuevo valor n1, y procede con paso m	RENUM 10, 5, 5 RENUM 300, 2, 12
EDIT n	Activa la función de editing en la línea número n	EDIT 10
LIST n, m	Presenta en pantalla la lista de instrucciones comprendidas entre los números n y m	LIST 5, 50
LLIST n, m	Realiza la misma función en la impresora	LLIST 3, 25
RUN	Pone en ejecución el programa residente en memoria	RUN
TRON	Activa la visualización (en pantalla) de la trayectoria del programa	TRON
TROFF	Desactiva las funciones activadas por la orden TRON	TROFF
MERGE "YYY"	Toma el programa YYY del disco y lo une con el residente en memoria	MERGE "PROG-1"

memorias masivas (discos o cintas) se enumeran por separado.

En la exposición de la sintaxis se ha optado a veces por insertar algunos espacios en blanco (blanks) entre las palabras clave, los nombres de las variables y los operadores, para describir mejor las distintas partes de una misma instrucción. Sin embargo, hay que señalar que el intérprete Basic (o el compilador) no atribuye significado alguno a los espacios que separan los distintos componentes de la instrucción, puesto que la reconoce secuencialmente en cualquier parte de la línea en que se encuentren (si está bien escrita y en el orden correcto).

Inicio y final de los programas

Un programa cualquiera ha de tener una instrucción de comienzo y una de final, del mismo modo que los diagramas de flujo se empiezan siempre con el símbolo START (comienzo) y se terminan siempre con END (fin).

Para algunos lenguajes, entre ellos el Basic, la instrucción de comienzo falta, puesto que el ordenador comienza siempre automáticamente la ejecución de un programa por la instrucción con el número de línea menor y sigue hacia el mayor. El automatismo que se obtiene aprovechando el número de líneas inicial no sirve para individualizar el fin del programa, puesto que las instrucciones de numeración más alta no siempre son las últimas en ser ejecutadas.

Normalmente, un programa está estructurado en dos partes: el programa principal (main) y las subrutinas o subprogramas. El principal llama las diversas subrutinas y les pasa el control. Al final de cada subrutina, la ejecución del programa vuelve al flujo principal. La ejecución se «salta» algunos números de línea para retomarlos posteriormente, por lo que no hay secuencialidad con respecto a estos números.

UTILIZACION DE LOS COMANDOS RUN, TRON, TROFF




```
LIST
10 B = 5
15 C = 28
50 A = B + C
80 D = A * 6
82 K = C - B
90 PRINT "  ***  RUN  END  *** "
95 END
Ok
TRON
Ok
RUN
[10][15][50][80][82][90]  ***  RUN  END  ***
[95]
Ok
TROFF
Ok
RUN
   ***  RUN  END  ***
Ok
```

En la primera ínea se pide el listado del programa actualmente residente en memoria.

Las líneas del programa aparecen en el monitor, y el listado termina con el mensaje de orden cumplida.

Con esta orden, el operador activa la visualización de los números de l'nea de las instrucciones que serán ejecutadas.

El comando RUN activa la ejecución del programa residente en memoria. Los números de las líneas ejecutadas aparecen en el monitor entre corchetes.

Esta parte de la línea se visualiza a causa de la instrucción de impresión contenida en la línea 90.

Tras el mensaje de comando cumplido, el operador desactiva el comando TRON con el nuevo comando TROFF.

Tras un nuevo requerimiento de ejecución, la única salida presentada es la llamada en la línea 90.

En la pág. 369 se esquematiza un programa que incluye dos subrutinas (A y B). La primera en ser llamada es la subrutina A; su numeración es del todo arbitraria. Puede hallarse en un punto cualquiera del programa, independientemente del hecho de ser llamada la primera, y puede tener, por ejemplo, una numeración comprendida entre 3700 y 4580. Análogamente, la subrutina B ocupa otra posición arbitraria y puede estar escrita antes que la A (en el gráfico está entre las líneas 1350 y 1920).

La elección de la secuencia de llamada tiene lugar en el programa principal, en las líneas 50 y 95. Al final de la primera subrutina (subrutina A) el programa contiene la instrucción RETURN (línea 4580). Este código devuelve el control al programa principal (instrucción 60) y la máquina prosigue con las instrucciones 60, 65, 95. En esta última línea se halla la llamada a la subrutina B; por tanto, la ejecución pasa a la línea 1350, y prosigue hacia el final de la subrutina B. La instrucción 1920 provoca el retorno al programa principal (instrucción 100). A partir de esta línea, la máquina procede consecutivamente (instrucciones 100, 110, etc.) y si falta la instrucción de fin de programa (END) vuelve a entrar en la subrutina B (sin que haya sido llamada nuevamente). En esta subrutina, a la instrucción 1350 (RETURN) no le corresponde ninguna dirección de retorno; de este modo se generan errores y el programa da resultados incorrectos.

Siguiendo con el diagrama de la pág. 369, la instrucción END se halla al final lógico de la ejecución, tras las dos subrutinas. En realidad, la posición «física» (es decir, el número de línea) está en el programa principal, antes de la subrutina A. El flujo correcto mostrado en el punto **c** constituye el detalle del diagrama en el punto **a**, y representa con símbolos gráficos el mecanismo ilustrado detalladamente en el punto **b**.

El inicio de la ejecución del programa tiene lugar automáticamente a partir de la instrucción con la numeración más baja, y por tanto, es en esta instrucción donde hay que especificar qué «base» se desea usar a lo largo del programa.

El ordenador utiliza para los índices también el valor 0, por lo que el primer elemento de una serie cualquiera es identificado como el número cero. Sin embargo, al usuario le resulta más cómodo indicar el primer elemento de una serie con el número 1. A tal objeto existe la instrucción OPTION BASE 1, que cambia la base utilizada por la máquina. Así, en una tabla de cinco valores, para el ordenador (en Basic) los elementos se ordenan como 0, 1, 2, 3, 4, mientras que para el programador es más cómodo indicarlos a partir de 1. Esta indicación se suministra al ordenador con la instrucción OPTION BASE 1. Tras leer esta instrucción, la máquina también utilizará los índices a partir de 1.

En realidad se cambia sólo la forma de presentación: especificando OPTION BASE 1, el sistema no hace más que restar 1 a los valores de los índices, pasándolos así a su propia simbología, que empieza por 0. Resumiendo: la primera instrucción de un programa puede ser OPTION BASE 1; la última (en el sentido lógico y no como numeración) ha de ser END.

A lo largo de un programa, a menudo es necesario escribir algunos comentarios (y en todo caso es conveniente hacerlo) para recordar las funciones realizadas y los métodos empleados. Si no incluye las explicaciones pertinentes, un programa que haya de ser reexaminado algún tiempo después se vuelve incomprensible. Es, por tanto, necesario indicarle al ordenador que una determinada línea no es una instrucción sino un simple comentario. Esta distinción se logra comenzando la línea de comentario con la sigla REM (abreviatura de la palabra inglesa remark, nota) o con un símbolo adecuado, normalmente una comilla (en algunas máquinas, un signo de exclamación). Ejemplo:

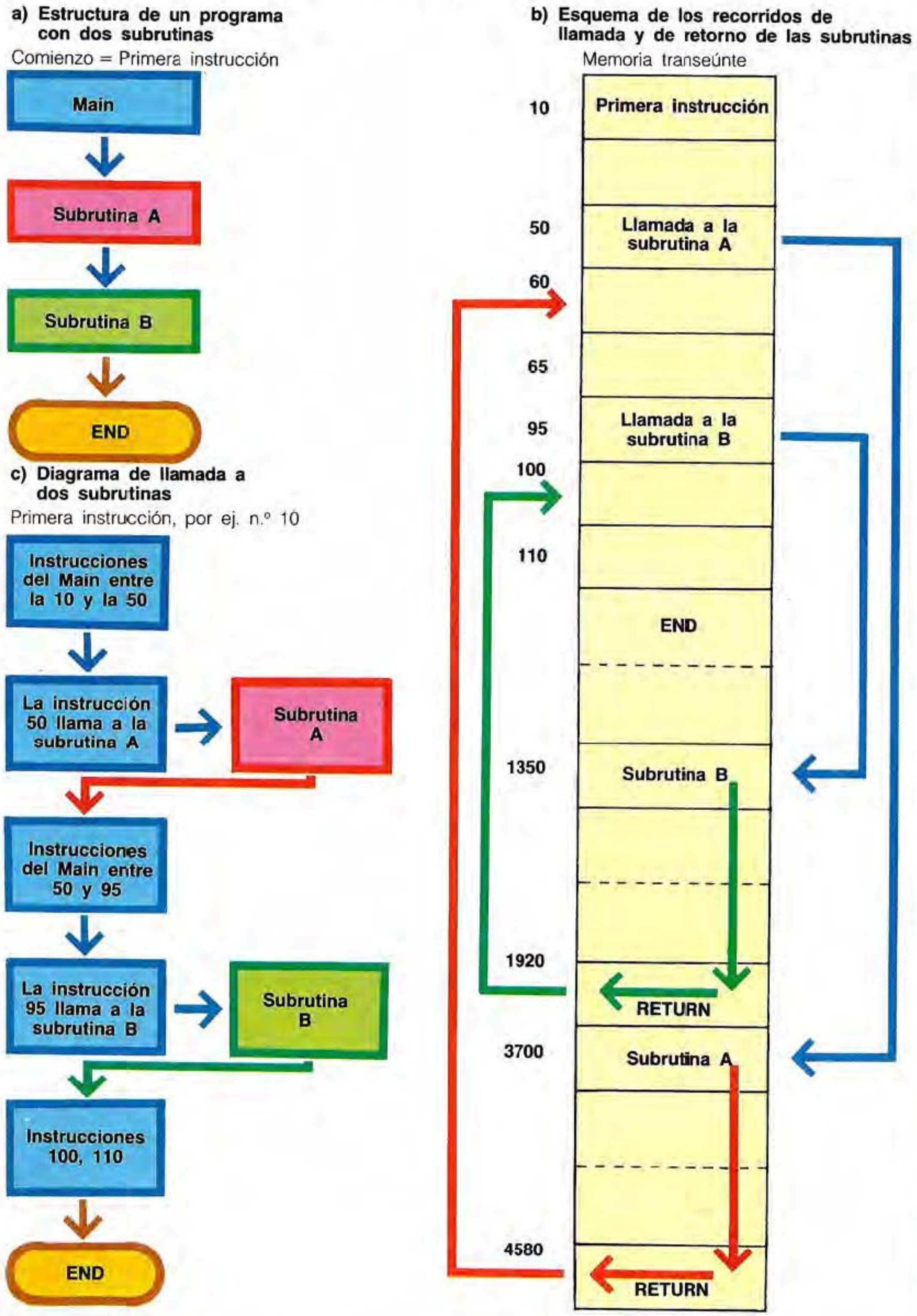
```
10 OPTION BASE 1
20 REM Esta línea es un comentario
30 A = B + C 'Suma
(30 A = B + C !Suma)
40 END
```

La línea 20 es un comentario indicado con el código REM, mientras que la 30 muestra el uso de los símbolos alternativos ' o !; todo lo que se halla a la derecha del símbolo se considera un comentario y no es elaborado. Los comentarios son interpretados como tales y no se ejecutan, pero también ocupan memoria.

Declaración del tipo de las variables

Las constantes y las variables que aparecen en un programa pueden pertenecer a uno de los cuatro tipos posibles (enteras, reales, doble precisión, cadenas). La declaración del tipo al que pertenecen tiene lugar poniendo a continuación del nombre o el valor el símbolo pertinente (% = entera, ! = real, # = doble precisión, $ = cadena).

ESTRUCTURA LOGICA Y ESQUEMA DE EJECUCION DE UN PROGRAMA



Con esta simbología, los nombres de las variables siempre van seguidos por el símbolo que especifica su tipo; sólo para las reales el símbolo ! puede ser omitido, puesto que, a falta de especificaciones contrarias, se da por supuesto (en general, los parámetros y las especificaciones asumidos automáticamente por la máquina a falta de indicaciones concretas del programador, se dicen asumidos por **default** o presunción por omisión).

En la escritura de los programas es muy incómodo escribir los nombres de las variables repitiendo cada vez el símbolo de su tipo. Se obtiene, además, un listado sobrecargado de símbolos y, por lo tanto, de difícil lectura. Como alternativa a la declaración «explícita» (el símbolo detrás del nombre), se puede utilizar la declaración «implícita», es decir, una instrucción concreta para especificar que una variable o un grupo de variables pertenecen a un tipo determinado. Tras la declaración ya no es necesario símbolo alguno y, a lo largo del programa, las variables así declaradas siempre se consideran como del tipo especificado. Las instrucciones de declaración implícita de clase son:

DEFINT	para los enteros
DEFSNG	reales en precisión simple
DEFDBL	reales en doble precisión
DEFSTR	cadenas

El símbolo DEF de estas instrucciones no debe confundirse con el DEF FN utilizado para definir las funciones; ambos indican a la máquina que la instrucción es una definición, pero con significados completamente distintos.

La definición tiene lugar indicando, después de la instrucción, la letra por la que empezarán los nombres de las variables de ese tipo.

Por ejemplo: DEFINT I-N indica que todos los nombres utilizados a lo largo del programa que comiencen por las letras comprendidas entre la I y la N (I, J, K, L, M y N) son variables enteras.

La definición puede contener varios grupos de letras separados por una coma, por ejemplo:

DEFINT I-N, X-Z	Los nombres que comienzan por letras comprendidas en los grupos I-N y X-Z indican variables enteras.
DEFDBL A-D, P-T	Las iniciales de las variables en doble precisión han de estar comprendidas en los intervalos A-D y P-T.

No en todos los sistemas existe la declaración implícita. En algunos hay que declarar el nombre completo de la variable, y no se puede definir un intervalo de letras; (para cada variable hay que escribir una declaración de tipo).

El empleo de la declaración implícita es muy indicado para declarar los reales en doble precisión y para los enteros. Los reales en simple precisión son asumidos como tales por default, por lo que es inútil declararlos (salvo como recordatorio).

La definición anterior (DEFINT I-N) puesta como ejemplo, es en realidad muy importante al objeto de una buena programación. En la escritura de un bucle (ver «Los bucles», pág. 174) se utiliza un índice que, partiendo de un valor inicial, llega tras sucesivos incrementos a un valor final, que interrumpe la ejecución del bucle mismo, o de partes recurrentes incluidas en él. Este índice ha de ser entero. Si se declara como real, se utiliza sólo su parte entera; no hay errores, pero se ocupa más memoria de la necesaria y se hace mucho más lenta la ejecución del bucle. La utilidad de esta precaución es tal que, en Fortran (lenguaje muy similar al Basic), el sistema operativo está estructurado de forma que todos los nombres que comienzan por una letra comprendida en el intervalo I-N designan siempre variables enteras, sin necesidad de declararlo. Por ello conviene utilizar la declaración DEFINT I-N y usar las letras de dicho intervalo como iniciales de los índices de los bucles.

Instrucciones de asignación

A este grupo pertenecen las instrucciones que asignan a las variables o a las constantes un determinado valor. Este valor puede ser una constante (numérica o cadena), una variable o el resultado de un cálculo.

Toda magnitud utilizada en un programa ha de tener un nombre simbólico al cual el sistema operativo asocia una determinada posición de memoria: utilizando el nombre simbólico asociado a la memoria, se toma su contenido o se escribe en ella un dato. En otras palabras, es como si el nombre de cada variable se convirtiera en una dirección de memoria en donde se deposita (y actualiza continuamente) el valor que va tomando dicha variable.

LET

La instrucción de asignación más simple es:

LET nombre = valor

De este modo se pide a la máquina que escriba el dato «valor» en la memoria llamada «nombre». Por ejemplo, la instrucción LET A = 2.5 pone el valor numérico 2.5 en una posición de memoria que de ahora en adelante se denominará A; a partir de esta instrucción se podrá usar el símbolo A como si fuera el valor numérico correspondiente.
En casi todos los sistemas, el código LET puede ser omitido. La instrucción A = 2.5 equivale a la anterior y, en este caso, es el símbolo = el que le indica a la máquina que ha de ejecutar una instrucción de asignación.
Usando el símbolo = se puede realizar un cálculo cualquiera y memorizar su resultado en una variable, que a su vez puede ser utilizada para efectuar nuevos cálculos o enviada a impresión.
La instrucción de impresión se describirá más adelante; de momento, para entendernos, utilizaremos la forma más simple: PRINT nombre.
Esta instrucción provoca la escritura en pantalla del valor ligado al símbolo «nombre»; en el ejemplo anterior (LET A = 2.5), para ver el resultado hace falta la instrucción PRINT A. En pantalla aparecerá el número 2.5.
Por ejemplo, desarrollemos el cálculo del área de un círculo de radio R = 3 cm. La fórmula a utilizar es Area = 3.14 × R^2, que sustituyendo R por su valor se convierte en: Area = 3.14 × 3^2.
El programa podría ser:

```
10' Falta OPTION BASE 1 puesto que no
15' se utilizan índices
20 AREA = 3.14 * 3Λ2 'el símbolo Λ indi-
25' ca elevación a potencia
30 PRINT AREA
40 END
RUN (orden de activación de la ejecución)
28.26 (resultado del cálculo)
```

Las líneas 10, 15 y 25 contienen comentarios (símbolo '); la 20 contiene tanto el cálculo como la asignación del resultado a la variable AREA; la 30 es la instrucción para la impresión de AREA (en pantalla); la 40 es el fin de programa.
Pueden escribirse más instrucciones en la misma línea utilizando el símbolo : (o /, según el sistema operativo). Usando este símbolo, el programa anterior puede reducirse a una sola línea (omitiendo los comentarios):

```
20 AREA = 3.14 * 3Λ2:PRINT AREA:END
RUN
28.26
```

Este programa no tiene utilidad práctica; si varía el radio del círculo hay que reescribir la instrucción, suministrando el nuevo valor del radio (en el ejemplo, R = 3). En las aplicaciones, se ha de hacer que el programa sea paramétrico con respecto a los datos.
Si el programa estuviera escrito en la forma siguiente:

```
10 R = 3
20 AREA = 3.14 * RΛ2
30 PRINT AREA
40 END
```

en la línea 20 el radio ya no se indicaría con su valor numérico sino con un símbolo, o sea una variable. Cambiando el valor numérico asignado a la variable R en la instrucción 10, el cálculo del área (línea 20) sería válido para cualquier círculo. También en esta forma existe la limitación debida a la línea 10: para variar el radio hay que modificar una instrucción, por lo que este programa tampoco es realmente utilizable. La única manera de generalizarlo, haciéndolo válido para cualquier valor de R, consiste en dejar al usuario la posibilidad de comunicar a la máquina dicho valor, que podrá ser diferente en cada ocasión.
La instrucción a utilizar, como veremos, es INPUT R. De esta manera la máquina se detiene en espera de la introducción, mediante teclado, del valor de R; en cuanto lo recibe, efectúa el cálculo y presenta el resultado.
Una forma más correcta de plantear el programa es, por tanto, la siguiente:

```
10 INPUT R
20 AREA = 3.14 * RΛ2
30 PRINT AREA
40 END
```

Durante el desarrollo del programa, el ordenador se detiene en la instrucción 10, escribe en pantalla el símbolo ? para informar al operador de que hace falta una introducción de datos y permanece a la espera. El operador digita en el teclado el valor de R; al terminar la digitación del dato, pulsa la tecla RETURN para informar al ordenador de que el dato está completo. El valor numérico introducido es asignado a la variable R, y en todas las instrucciones en las que aparece este símbolo es como si estuviera escrito el valor introducido por teclado.

En la línea 20 se utiliza el símbolo Λ, que significa potencia (en este caso concreto, la potencia 2, o sea R^2), pero no todas las máquinas permiten la ejecución directa de esta operación.
De todos modos, el cálculo puede realizarse multiplicando R por sí mismo ($R^2 = R \times R$) y la 20 se convierte en: AREA = 3.14 * R * R.
En el listado de la página contigua vemos una parte de programa que, en función de los parámetros de entrada, calcula el área y el perímetro de algunas figuras planas.
El tipo de figura es indicado por el valor del flag K. Según el valor de este flag (1, 2, etc.) habrá que seleccionar la fórmula adecuada (líneas 1670, 1700, etc.; la parte que efectúa la selección será mostrada más adelante).
Cada cálculo, comprendida la instrucción de retorno al programa principal (main), se desarrolla en una sola línea, utilizando el símbolo de continuación:. Todas las líneas que (tras la numeración) contienen el símbolo ' (1500, 1510, etc.) son comentarios.
Las entradas a suministrar son los valores de los datos (los que sean necesarios en función de cada figura geométrica) y el tipo de figura a seleccionar (K). En salida, se tiene el área en la variable denominada AREA y el perímetro en la variable PER.
Por ejemplo, poniendo K = 5 (círculo) y LADO1 = 7 (en el círculo, LADO1 significa el radio, y los demás lados no sirven) se tiene: AREA = 153.86 y PER = 43.96 (con el significado de longitud de la circunferencia).
Si llamáramos la rutina con K = 3 (trapecio), tendríamos que dar los valores de los cuatro lados, puesto que todos son necesarios. Obsérvese que en el listado no aparece la instrucción END, puesto que se trata de una subrutina; la instrucción de fin de programa se halla en el programa principal que utiliza esta subrutina.
La instrucción de asignación con el código operativo LET casi nunca se utiliza: normalmente se emplea la forma implícita (con el símbolo =; por ejemplo, A = 21 en lugar de LET A = 21); esta forma constituye el único ejemplo de instrucción que no comienza con una «palabra reservada», es decir, con una palabra conocida por el sistema operativo que designa la acción a realizar.
Para todas las demás instrucciones, tras el número de línea ha de ir el código.
Los vocablos que designan los códigos operativos (acciones a realizar) están reservados al sistema, y en los programas no pueden usarse como nombres de variables. Por ejemplo, si en el programa para calcular las áreas adoptáramos para el perímetro el nombre PRINT (en lugar de PER) tendríamos error, puesto que el sistema reconoce un nombre reservado y no lo asigna a una variable.
Algunos intérpretes Basic pueden distinguir, analizando el contenido de la frase, si la palabra se refiere a una instrucción o a un nombre de variable, pero en general es conveniente no caer en esta ambigüedad evitando los nombres reservados.

DATA, READ, RESTORE

Las instrucciones DATA y READ han de usarse conjuntamente, aunque sea en zonas separadas del programa, y constituyen otro medio para asignar valores a una o más variables.
El código DATA indica a la máquina que, tras la instrucción, vendrán una serie de números o letras a interpretar como datos para algunas variables. Las variables a las cuales se han de asignar se indican con la instrucción READ.
Por ejemplo, las instrucciones:

```
10 READ X
20 DATA 3.5
```

asignan a X el valor 3.5, igual que la instrucción X = 3.5.
Las diferencias entre la instrucción DATA y la mera asignación se pueden resumir así:

— con una sola instrucción DATA se pueden asignar varios valores a varias variables;
— los valores contenidos en DATA pueden ser utilizados también varias veces para diversas variables.

La instrucción DATA permite asignaciones múltiples. Por ejemplo, si hay que asignar los valores 2, 7.3, 15 y 174 respectivamente a las variables R, VAL, NOMBRE y ESTADO, con la primera forma (LET implícito) la instrucción es:

```
10 R = 2 : VAL = 7.3 : NOMBRE = 15 : ESTADO = 174
```

y los valores numéricos 2, 7.3, etc. no pueden volver a utilizarse.
Si a lo largo del programa una nueva variable, por ejemplo T, ha de tomar el valor 2, deberá reescribirse la instrucción correspondiente (T = 2), sin poder volver a utilizar el valor 2 escrito en la instrucción número 10. Por el contrario, utili-

SUBRUTINAS PARA EL CALCULO DE AREAS Y PERIMETROS

```
1500'   **Subrutinas para el cálculo de Areas y Perímetros
1510'
1520'      SIMBOLOS USADOS:
1530'                         AREA = Valor del área calculada
1540'                         PER  = Valor del perímetro
1550'                         LADO1= Valor en Entrada de uno de los lados
1560'                         LADO2= Valor en Entrada del segundo lado
1570'                         LADO3= Si es necesario
1580'                         LADO4= Si es necesario
1590'  K = FLAG que indica el tipo de figura con los valores
1600'                            K = 1   CUADRADO
1610'                            K = 2   RECTANGULO
1620'                            K = 3   TRAPECIO
1630'                            K = 4   ROMBO
1640'                            K = 5   CIRCULO
1650'
1660'  Cuadrado (K=1)
1670 AREA=LADO1*LADO1:PER=4*LADO1:RETURN
1680'  Rectángulo (K=2)
1690'  LADO1=Base     LADO2=Altura
1700 AREA=LADO1*LADO2:PER=2*(LADO1 + LADO2):RETURN
1710'  Trapecio (K=3)
1720'  LADO1=Base menor LADO2=Base mayor LADO3 y LADO4=Lados Oblicuos ALT=Altura
1730 AREA=(LADO1+LADO2)*ALT/2:PER =LADO1+LADO2+LADO3+LADO4:RETURN
1740'  ROMBO (K=4)
1750'  LADO1=Diagonal menor    LADO2=Diagonal mayor    LADO3=Lado
1760 AREA=LADO1*LADO2/2:PER=4*LADO3:RETURN
1770'  Círculo (K=5)
1780'  LADO1=Radio
1790 AREA=3.14*(LADO1*LADO1):PER=6.28*LADO1:RETURN
1800'
```

SOLUCIONES DEL TEST 10

1 / a: la versión ASCII es la de nombre TEST, memorizada en el disco de la unidad A.
b: el programa PRUEBA 2 (de tipo protegido, P = Protegido) se halla en el disco que en el momento de la introducción de la orden estaba en la unidad B.
c: las correcciones pueden efectuarse en las versiones de nombre PRUEBA y TEST.

---

2 / Los valores de las variables D y E se obtienen sustituyendo en las líneas 40 y 50 los valores de A, B y C asignados en las líneas 10, 20 y 30:

$$D = 10 \times (6 + 8) = 140$$

$$E = 2 \times \frac{140}{6} = 46.666... = 46.\bar{6}$$

La simbología $46.\bar{6}$ indica que la cifra $\bar{6}$ (período) se repite hasta el infinito. Este resultado, exacto desde el punto de vista matemático, es aproximado por el ordenador al número de decimales característico de la máquina, que determina su precisión. Volviendo a numerar el programa a partir de 20 y con paso 2, tenemos:

```
20 A = 6
22 B = 8
24 C = 10
26 D = C * (A + B)
28 E = 2 * D/A
```

Uniendo este programa con:

```
20 B = 2
30 F = D + E
```

la anterior línea 20 A = 6 es sustituida por la 20 B = 2, por lo que el programa se convierte en:

```
20 B = 2
22 B = 8
24 C = 10
26 D = C * (A + B)
28 E = 2 * D/A
```

A la variable B se le asignan consecutivamente dos valores distintos (líneas 20 y 22); de los dos, se conserva el último, por lo que en los cálculos siguientes el valor de B es 8. La variable A ya no está inicializada. En estos casos, el Basic asigna el valor 0; por lo tanto, en las líneas 26 y 28 se tiene A = 0. La línea 26 da:

$$D = 10 * 8 = 80 \quad (A = 0)$$

mientras que la línea 28 causa un error, ya que se tiene una división por 0:

$$E = 2 \times \frac{80}{0} \quad (D = 80,\ A = 0)$$

---

3 / El comando REN 10,10,2 hace que se vuelvan a numerar las líneas 10, 20 y 30, obteniendo:

```
10 A = 3
12 B = 2
14 C = 3 * (A + B)
```

El comando AUTO 20,5 genera la numeración automática a partir de 20 y con paso 5. Las líneas 20 F = 2 * (C + A) y 25 PRINT F son tomadas tal como están y añadidas a continuación de las anteriores. Por lo tanto, el programa resultante es el siguiente:

```
10 A = 3
12 B = 2
14 C = 3 * (A + B)
20 F = 2 * (C + A)
25 PRINT F
```

El hecho de que falten algunos números de línea (por ejemplo, los comprendidos entre 14 y 20) no es significativo, puesto que la numeración de las instrucciones es del todo arbitraria (basta que sea progresiva).

---

4 / Hay dos maneras de hacerlo:
a) digitar una nueva línea 20 completa;
b) operar en Editor.
En el caso a) basta con digitar al final del programa la nueva línea 20 F = 4 * (C + A), que se superpondrá a la antigua línea 20. En el caso b) hay que pasar a la modalidad Editor introduciendo el comando EDIT 20. El sistema se posiciona entonces sobre la antigua línea 20 F = 2 * (C + A). La sustitución requerida se puede lograr cambiando el carácter 2 por el nuevo carácter 4.

---

5 / La instrucción 20 TRON activa el proceso de visualización de los números de las instrucciones ejecutadas. La línea 50 desactiva esta opción. Todas las instrucciones comprendidas (y por tanto ejecutadas) entre los dos comandos se visualizan en pantalla. En este caso concreto la salida es:

[30] [40]

puesto que entre el comando TRON y el comando TROFF sólo son ejecutadas las líneas 30 y 40. El comando de memorización es:

SAVE "A:PRUEBA",A

---

zando la instrucción DATA, los valores introducidos son de uso común para todas las variables, y pueden usarse varias veces.
Utilizando READ, la instrucción de asignación anterior se convierte en:

```
10 READ R, VAL, NOMBRE, ESTADO
.............................................
.............................................
273 DATA 2, 7.3, 15, 174
```

La instrucción DATA puede hallarse en cualquier punto del programa, aunque en algunas máquinas ha de ponerse al final del mismo, ya que no representa una instrucción ejecutable, sino sólo un área de memoria en la que se escriben algunos valores (que pueden ser literales); si se la encontrara a lo largo del programa, la máquina podría caer en error. La asignación de los valores se realiza en el orden progresivo en que están escritos: así, en la línea 10, el primer valor (2) se asigna a la primera variable (R), el segundo a la segunda (VAL), etc.
La instrucción READ puede dividirse en varias partes, cada una con algunas de las variables. El sistema, al ejecutar las partes de la instrucción, tiene en cuenta los valores ya asignados y procede siempre en orden progresivo.

Andante con bit para ordenador solo

Con justo orgullo, a los lados de uno de los stands de la Musikmesse de Frankfurt destacaba el letrero: «From the men who started it all». El stand era el de la Moog Company. Todo, o casi todo, enpezó, efectivamente, con Robert Moog, cuya innovación nos parece hoy día de una desconcertante sencillez: controlar mediante tensiones, en vez de operaciones manuales, dispositivos electrónicos para la generación y manipulación de sonidos (osciladores, filtros, amplificadores, etc.). Desde ese momento, procesos que antes requerían horas o jornadas de trabajo en un estudio de grabación se pudieron realizar en tiempos mucho más breves (directamente en tiempo real), permitiendo la construcción de los primeros sintetizadores, aparatos que podían usarse también para la ejecución en vivo, sobre el escenario.

Los sintetizadores son los instrumentos electrónicos por excelencia, aquellos en los que mayor es el control del usuario sobre la producción y la manipulación de los sonidos. Son también, por esto mismo, los instrumentos menos fáciles de usar, puesto que cabe descubrir casualmente alguna combinación interesante, pero es casi imposible llegar a una meta prefijada mediante tanteos, sin tener muy claro el funcionamiento global. Para usar bien un sintetizador, es importante tener al menos algunas nociones básicas de acústica, así como de las propiedades fundamentales del sonido, las formas de onda, y el funcionamiento de los distintos módulos (osciladores, filtros, amplificadores, moduladores, etc.) que constituyen el aparato.

A diferencia de un órgano electrónico, un sintetizador no está provisto necesariamente de un teclado similar al de un piano: la mayor parte de los modelos tienen teclado porque sigue siendo, por decirlo así, el periférico de entrada más cómodo para un músico que se ha formado con los instrumentos tradicionales; pero es sólo uno de los varios dispositivos de entrada. Hay en el mercado sistemas modulares (Moog y Roland, sobre todo) perfectamente utilizables sin teclado, y el VCS3 (famoso por el uso que de él han hecho Pink Floyd, Klaus Schulze y otros músicos) disponía de un teclado como accesorio, pero constituía de por sí un pequeño estudio de música electrónica.

*En lugar de teclado, por ejemplo, un sintetizador puede utilizar como periférico de entrada un s*ecuenciador *o un controlador de cinta. Es un dispositivo bastante difundido: en esencia, se trata de un instrumento que memoriza secuencias de tensiones de control (en el caso analógico) o de datos codificados digitalmente, que, por tanto, puede incluso repetir de forma cíclica. En el caso más típico se memorizan instrucciones que corresponden a la generación de notas concretas (con sus timbres concretos): el secuenciador gobierna luego el sintetizador de manera que reproduzca las notas programadas en la secuencia establecida (a una velocidad que puede ser controlada y variada).*

También se pueden conectar el secuenciador y el teclado y programar el primero con el segundo, tocando una vez la sucesión de notas deseada, que luego el secuenciador reproducirá fielmente (o a otra velocidad cualquiera), mientras que con el teclado se podrá ejecutar simultáneamente otra parte de la melodía. Con una máquina digital es fácil resolver el problema de la memorización: los datos pueden almacenarse en memorias de trabajo o de masa análogas a las de un ordenador (RAM, fichas magnéticas, cassettes). Un instrumento puede tener, por ejemplo, dieciséis memorias de trabajo: en cada una de ellas se puede memorizar una regulación global de todos los elementos de la máquina, es decir, un timbre determinado. Basta pulsar un botón para pasar rápidamente de un timbre a otro. En general, estas memorias se conservan con el instrumento apagado, pero en los aparatos mejores sus contenidos pueden archivarse también en ficha o cassette, mediante una simple grabadora; por lo tanto, un músico puede programar todas las combinaciones que le interesan, archivarlas y volverlas a utilizar cuando desee. Los datos son de tipo digital, por lo que el procedimiento posee el mismo grado de precisión y seguridad que los sistemas de memorización de un ordenador, de los que en nada se diferencia.

Los instrumentos más sofisticados, con secuenciador y batería electrónica, se aproximan mucho al ideal de la one man band*: el conjunto formado por un solo intérprete; el secuenciador puede memorizar incluso miles de notas y ser polifónico, y puede estar sincronizado con la batería electrónica; mientras el secuenciador y la batería repiten lo que han memorizado (a una velocidad que puede variarse a voluntad), el in-*

térprete puede añadir a esta base nuevas líneas, una melodía, una improvisación.
El efecto global, con una buena amplificación, puede resultar casi increíble.
El Poly-61 de Korg es un modelo más avanzado del Poly-six aparecido en el mercado hace algún tiempo, y posee doce osciladores controlados digitalmente que permiten la generación de seis voces (es decir, se pueden tocar simultáneamente en el teclado hasta seis notas); tiene un interfaz para grabar en cassette los programas y poder recuperarlos, y posee varios automatismos, entre ellos la posibilidad de «sostener» un sonido incluso después de haber apartado los dedos del teclado, y un sistema que memoriza y repite arpegios y acordes, sincronizable con un secuenciador o una batería electrónica externa. En la memoria puede haber simultáneamente 64 programas.
Los modelos DX Yamaha presentan una original forma de generación de sonidos, de tipo digital, denominada «síntesis digital de modulación de frecuencia mediante algoritmos programables», y ofrecen también la posibilidad del «toque dinámico»: el teclado reacciona de forma distinta según sea mayor o menor la fuerza con que se pulsan las teclas, como ocurre en un piano; además, este tipo de teclado permanece sensible a las variaciones de presión incluso después de la pulsación, es decir, permite iniciar una nota a baja intensidad y aumentar luego su volumen haciendo más presión sobre la tecla.
Otro entre los sintetizadores recientes más interesantes es el Chroma, proyectado por Arp (una de las primeras grandes empresas dedicadas a la fabricación de estos instrumentos). Se trata de un sintetizador polifónico completamente programable, fabricado con una tecnología mixta analógico-digital, provisto de dos microprocesadores: el principal (con 7K de RAM y 16K de ROM) dedicado a todos los controles de sonido, y el secundario, al control de la sensibilidad del toque (mide la velocidad de descenso de una tecla pulsada con una precisión de una milésima de segundo y, en función de los datos así obtenidos, permite la simulación del toque como en un teclado de piano). En la RAM del microprocesador principal pueden almacenarse hasta 50 programas simultáneamente, seleccionables desde un panel; los programas pueden conservarse también con el sintetizador apagado, gracias a un sistema de back-up a batería, y pueden archivarse en cassette. La firma suministra, de entrada, una cassette con 150 programas.
El campo más fascinante, en el que casi todo está aún por descubrir, es el de la computer music y de los instrumentos que utilizan un ordenador, especialmente uno personal. Muchos de los ordenadores personales que hay en el mercado disponen de osciladores internos, con los que se pueden realizar melodías o efectos sonoros; pero las posibilidades se amplían extraordinariamente cuando el ordenador se potencia con unidades de expansión especiales o se conecta con un sintetizador y puede oficiar como centro de control de todas sus regulaciones y actividades. Concretamente, para el Apple se han desarrollado unidades hardware y algunos programas de notable interés, como el Music System, o el sistema Alpha Syntauri, basado en dos unidades producidas por la Mountain Computer Music System, que se comporta como un sintetizador digital polifónico de ocho voces, con la posibilidad de crear timbres por síntesis adicional o, simplemente, dibujando en la pantalla la forma de onda deseada; está provisto, además, de un teclado con control de la sensibilidad de toque.
Entre las novedades cabe mencionar un programa para la composición musical estudiado para el Apple II (pero pronto se pondrán a punto versiones para otros ordenadores personales, como el de IBM) de Amdek, que gobierna un periférico especial (el CMU-800, derivado del

Gran sintetizador por modulación de tensión, con memoria digital y mecanismo «track tape».



© BBC



El conjunto «Yes» grabó, en 1972, uno de los primeros álbumes de música rock ejecutada con ayuda de sintetizador: el revolucionario «Close to the edge».

Microcomposer de Roland) con el que se pueden componer fragmentos musicales y arreglos, así como llegar a controlar hasta ocho sintetizadores externos.

En niveles de precio netamente superiores (del orden de millones), las máquinas muy sofisticadas están reservadas aún, evidentemente, a los grandes estudios o los músicos famosos. Pero la evolución de la tecnología ha sido rápida y promete serlo aún más en los próximos años; por lo tanto, estas máquinas dan una idea de las características de los instrumentos que serán asequibles dentro de unos años. Sólo un ejemplo, a título meramente ilustrativo: el Wawe 2.2 y el Waweterm de PPG. El Wawe 2.2 es un sintetizador cuyo núcleo está formado por 16 osciladores y 30 tablas de formas de onda (64 formas por tabla, con un total de unas 2.000 formas de onda complejas) producidas digitalmente, mezclables, filtrables, con un microprocesador y un panel de control en parte digital y en parte analógico para satisfacer todas las exigencias y hacer más intuitivas, para su utilización en tiempo real, las diversas regulaciones posibles. El Digital Recording System permite la memorización de ocho pistas de secuencias de sonidos o de líneas melódicas introducidas con el teclado, y luego, durante el playback, *regulables independientemente. Ya de por sí complejo, el Wawe 2.2 se convierte en un aparato de grandes prestaciones una vez conectado con el Waweterm, un terminal de vídeo con capacidad gráfica, memoria de masa en floppy de 8 pulgadas y un elaborador de señales audio que acepta en entrada sonidos cualesquiera, captados mediante micrófono, los analiza y da su forma de onda en pantalla. Diez teclas justo debajo de la pantalla permiten efectuar un* editing *de la forma visualizada, que luego puede enviarse al Wawe 2.2 y utilizarse como una de sus muchas formas de onda. Esto significa que, si se graba una voz humana, se la podrá reproducir mediante el teclado, desplazándola hacia las diversas frecuencias y obteniendo así voces de bajo, de tenor, de soprano, etc., cualquiera que fuese la voz inicialmente grabada. El grado de correspondencia con el original es impresionante. Pero, obviamente, el Wawe 2.2 no se limita a reproducir fielmente, sino que puede tratar por ordenador la forma de onda recibida; se puede, pues, manipularla mediante los filtros y los generadores de envolvente y mezclarla con otras formas de onda, obteniendo los resultados más sorprendentes.*

(Virginio Sala. Extracto de «Sette note digitali», en la publicación italiana ZEROUNO, n.° 15, abril de 1983.)

Por ejemplo, la instrucción 10 puede dividirse de la forma siguiente:

```
10 READ R, VAL
11 READ NOMBRE
12 READ ESTADO
...
273 DATA 2,7.3,15,174
```

En la 10 se asignan los valores R = 3 y VAL = 7.3, en la 11 NOMBRE = 15 y en la 12 ESTADO = 174.

El mecanismo de asignación de valores se ilustra en el gráfico inferior. A nivel de sistema, al DATA se le asocia un puntero (indicado con P en el gráfico) que indica cuál es el valor a tomar. Inicialmente P se pone igual a 1 e indica el acceso al primer valor de DATA; a cada lectura se incrementa en 1 e indica («apunta») el siguiente valor a tomar.

En el gráfico, la primera operación READ atañe a dos variables. A la primera se le asocia el valor numérico 2 (el puntero vale 1) y al final de esta asignación el puntero se incrementa (P = 2); a la segunda variable se le asocia el nuevo valor apuntado por P (7.3), y así sucesivamente. La última lectura (instrucción 12) lleva el puntero al valor 4, que indica el último valor de DATA. Ulteriores operaciones de lectura causarían error, ya que P ya ha alcanzado su valor máximo y no puede proseguir. Al volver a usar los valores contenidos en DATA hay que poner a cero el puntero P, devolviéndolo a su valor inicial*.

La instrucción que permite restablecer el valor inicial del puntero es RESTORE.

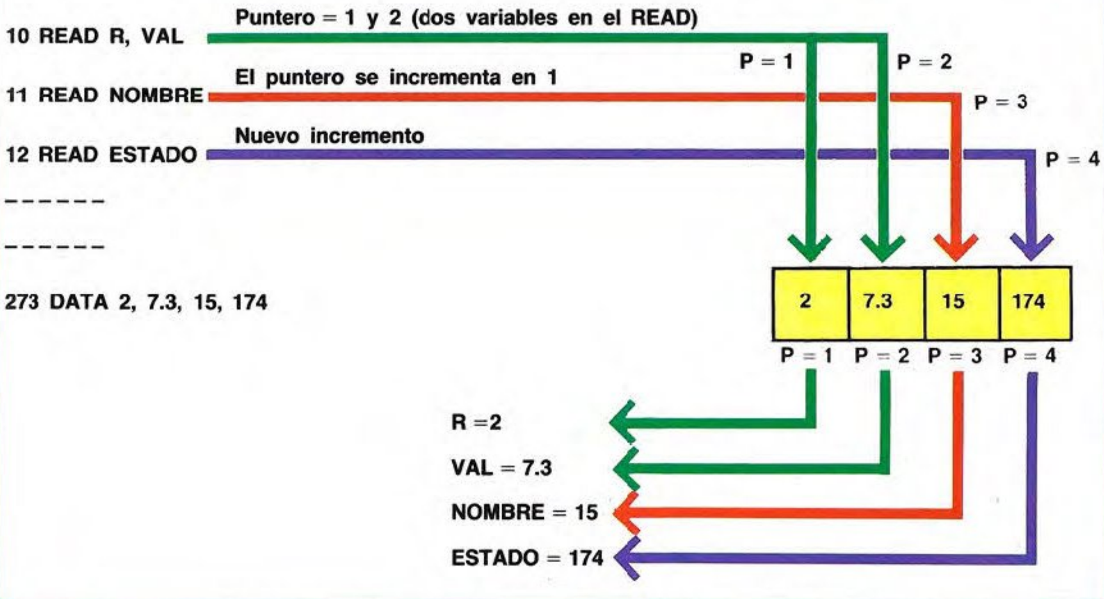
Tras esta instrucción, insertable en cualquier punto del programa, se vuelven accesibles a partir del primer valor todos los DATA presentes.

En la pág. 380, arriba, hay un ejemplo del uso de RESTORE. En las líneas 20 y 30, las operaciones de lectura para cinco variables (A, B, C, X e Y) desplazan el puntero de DATA del primer valor (8) al quinto (25); el siguiente valor tomado con una instrucción READ sería el dato 73. En la línea 50, el código RESTORE devuelve al puntero a su primer valor y, por tanto, vuelve accesible el DATA desde su comienzo, y el primer valor asignado con el siguiente READ (línea 60) es 8. Seguidamente son asignados los valores 19 y 11 y, por tanto, las variables E, F y G tomarán los primeros valores de DATA. Eliminando la instrucción 50, el puntero ya no es devuelto al valor inicial, y la siguiente instrucción READ (línea 60) accede al DATA a partir del mismo valor. Las instrucciones PRINT (líneas 40 y 70) muestran el contenido de las variables en ambos casos.

---

* El nombre P dado al puntero y los valores que toma son meramente indicativos y se utilizan sólo para explicar el mecanismo de ejecución de la asignación con DATA, cuya gestión corre a cargo del intérprete Basic, y por tanto es «transparente» para el programador.

ESQUEMA LOGICO DEL MECANISMO DE LOS PUNTEROS EN LOS DATA



USO DE LA INSTRUCCION RESTORE (1)

```
10 OPTION BASE 1
20 READ A,B,C
30 READ X,Y
40 LPRINT "A = " ;A,"B = "; B, "C = "; C," X = ";X,"Y = "; Y
50 RESTORE
60 READ E,F,G
70 LPRINT "E = ";E, "F = ";F, "G = "; G
80 DATA 8,19,11,2.7,25,73,80,9.1
```

```
A = 8          B = 19          C = 11          X = 2.7          Y = 25
E = 8          F = 19          G = 11

QUITANDO LA LINEA 50 SE OBTIENE:

A = 8          B = 19          C = 11          X = 2.7          Y = 25
E = 73         F = 80          G = 9.1
```

USO DE LA INSTRUCCION RESTORE (2)

```
10 OPTION BASE 1
20 READ A,B,C
30 LPRINT "A = "; A, "B = ";B,"C = ";C
40 READ X,Y
50 RESTORE 90
60 READ K,Z
70 LPRINT "X = ";X,"Y = ";Y,"K = ";K,"Z = ";Z
80 DATA 7,9,200
90 DATA 26,48,11,30
```

```
A = 7          B = 9          C = 200
X = 26         Y = 48         K = 26          Z = 48

SUSTITUYENDO LA LINEA 50 CON RESTORE SE OBTIENE:

A = 7          B = 9          C = 200
X = 26         Y = 48         K = 7           Z = 9
```

La instrucción RESTORE puede direccionarse a una línea concreta del DATA. En tal caso se vuelven de nuevo accesibles los valores a partir del DATA contenido en la línea, mientras que los DATA anteriores (es decir, con un número de línea inferior) no se vuelven a utilizar. En el listado superior, la instrucción RESTORE 90 (línea 50) indica que se desea acceder nuevamente a los valores contenidos en el DATA de la línea 90. La línea 40 asigna los valores X = 26, Y = 48 y desplaza el puntero al tercer valor (11). La operación RESTORE 90 devuelve el puntero al comienzo del DATA, y la siguiente READ (línea 60) toma nuevamente los dos primeros valores

ERROR EN EL USO DE LAS INSTRUCCIONES READ Y DATA

```
10 OPTION BASE 1
20 READ A,B
30 X=A+B
40 READ C,D,F
50 PRINT X,C,D,F
60 DATA 3,7,9,11



Out of DATA in 40
```

USO DE LA INSTRUCCION DATA PARA VARIABLES DE CADENA

```
10 OPTION BASE 1
20 READ A$
30 READ B$
40 READ C$
50 READ D$
60 READ E$
70 F$=A$+C$+B$+C$
80 G$=D$+C$+C$
90 H$=E$+C$
100 PRINT F$
110 PRINT G$
120 PRINT H$
130 L$=F$+G$+H$
140 PRINT L$
150 DATA "Apellido"
160 DATA "Nombre"
170 DATA "........."
180 DATA "Calle"
190 DATA "Ciudad"



Apellido...............Nombre................
Calle...................................
Ciudad..................................
Apellido...............Nombre................Calle...............Ciudad
```

(K = 26, Z = 48). Quitando la indicación de línea de la instrucción RESTORE 90, el puntero se posiciona en el primer DATA que encuentra tras el RESTORE (línea 80) y en este caso la línea 60 vuelve a asignar los valores 7 y 9 (en vez de 26 y 48).

El número de datos contenidos en las instrucciones DATA ha de ser al menos igual al número de variables que hay que leer (si es mayor, los datos excedentes son ignorados), de lo contrario el programa se detiene por error. La tabla superior muestra un programa que intenta leer cinco variables (líneas 20 y 40) en un DATA que contiene cuatro valores; el resultado es el men-

saje OUT OF DATA y el programa se interrumpe. La instrucción DATA puede utilizarse también para las variables de cadena. En este caso el contenido del DATA ha de especificarse entre comillas (el símbolo " indica el comienzo y el fin de una cadena). En el segundo listado de la pág. 381 se muestra un programa que utiliza cadenas. Las instrucciones de la 20 a la 60 cargan en las cadenas respectivas el contenido de los DATA. Con las operaciones de suma (instrucciones 70 y 80) se combinan oportunamente los datos insertando una serie de puntos en las posiciones deseadas (la cadena C$ contiene diez veces el símbolo ".") y se crean tres cadenas (F$, G$, H$) para la impresión de módulos de agenda. La impresión puede tener lugar sobre una o varias líneas, según como se combinen entre sí las diversas cadenas.

LSET y RSET

Estas instrucciones sirven para desplazar una constante o una variable de cadena alineándola a la izquierda, con LSET, o a la derecha, con RSET (L = Left, izquierda, R = Right, derecha). Por ejemplo, con la instrucción:

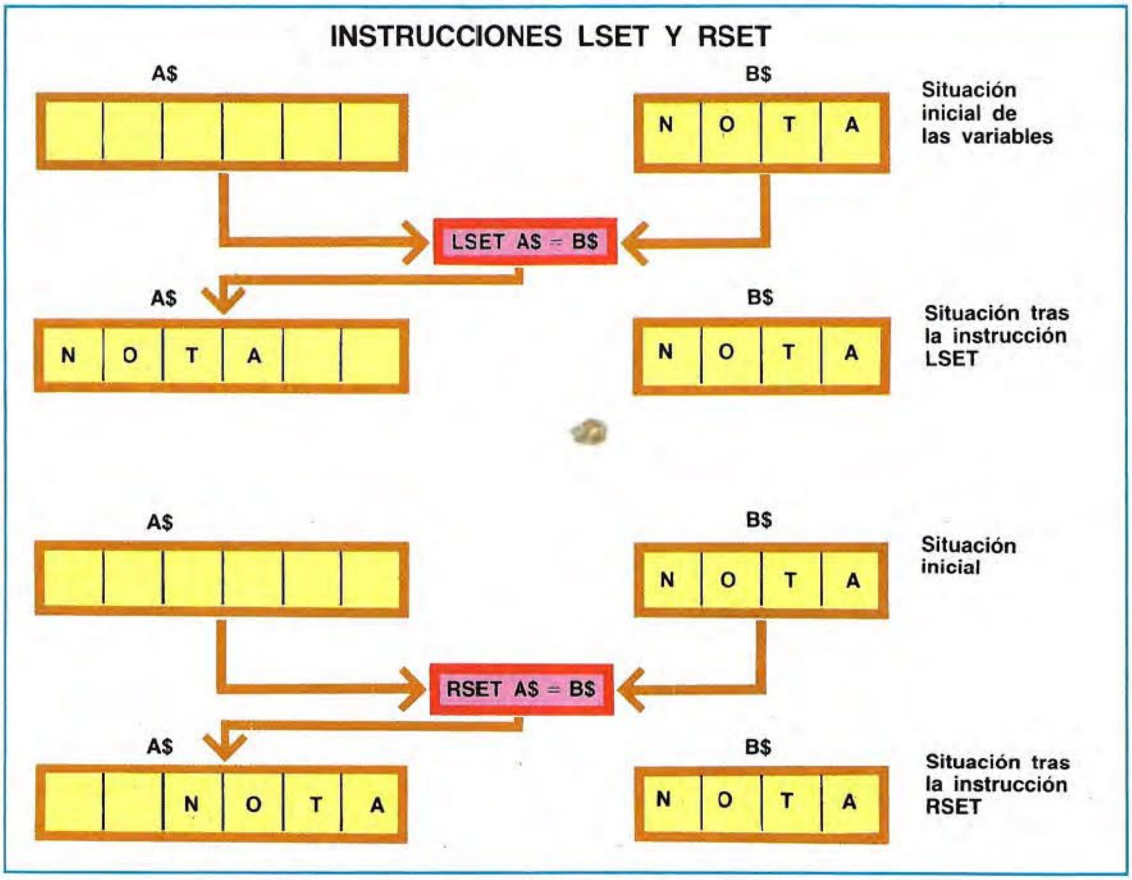
LSET A$ = B$

el contenido de B$ se transfiere a A$ poniéndolo a la izquierda, mientras que el contenido de B$ permanece invariable. Estas instrucciones se usan principalmente en la preparación de datos para imprimir o memorizar en disco.

En fase de impresión, aunque hay instrucciones específicas, es cómodo tener los campos ya alineados. Para el disco puede ser una condición esencial, ya que muchos sistemas memorizan sólo datos en ASCII (cadenas) y para transferir valores numéricos antes hay que convertirlos en cadenas. Entonces es necesario alinear los campos para poder obtener de nuevo los valores correctos en la conversión inversa (de caracteres a números, tras la lectura en disco).

SPACE$(N)

Esta instrucción (en realidad se trata de una función) asigna a una cadena la longitud (en caracteres) especificada por el valor de N y la llena de espacios en blanco (código hexadecimal 40; ver tabla ASCII). Por ejemplo, A$ = SPACE$(10)



USO DE LAS INSTRUCCIONES SPACE$, LSET, RSET

```
1000 '
1010 ' ** Uso de las instrucciones: SPACE$, LSET, RSET
1020 '
1030  Asigna a las variables NOMBRE1$, NOMBRE2$, NOMBRE3$, NOMBRE4$
1040 ' una longitud de 20 caracteres y las iniciales con espacios en blanco
1050 NOMBRE1$=SPACE$(20):NOMBRE2$=SPACE$(20):NOMBRE3$=SPACE$(20):NOMBRE4$=SPACE (20)
1060 '
1070 ' Lectura de 4 nombres. Las 4 instrucciones (de la 1090 a la 1120) pueden
1080 ' escribirse en una sola línea usando el símbolo ":"
1090 ' INPUT A1$ 'Esta instrucción lee en teclado lo que se introduce
1100 ' INPUT A2$ ' y lo transfiere a la variable especificada (A1$,A2$,A3$,A4$)
1110 ' INPUT A3$
1120 ' INPUT A4$
1130 '
1140 ' Alineación a la izquierda de los valores leídos en las variables A1$,...A4$
1150 ' en cada línea se ponen dos variables, pero pueden escribirse todas
1160 ' en la misma o cada una en una línea por separado
1170   LSET NOMBRE1$=A1$:LSET NOMBRE2$=A2$
1180   LSET NOMBRE3$=A3$:LSET NOMBRE4$=A4$
1190 '
1200 ' Impresión de los datos tal como se introducen (A1$....A4$)
1210 '
1220 PRINT A1$: PRINT A2$: PRINT A3$: PRINT A4$
1230 PRINT: PRINT ' Esta instrucción sirve para separar los resultados
1240 ' Impresión de los datos alineados a la iquierda (NOMBRE 1$... etc.)
1250 '
1260 PRINT NOMBRE1$: PRINT NOMBRE2$: PRINT NOMBRE3$: PRINT NOMBRE4$
1270 PRINT: PRINT ' Esta instrucción sirve para separar los resultados
1280 ' Alineación a la derecha
1290 '
1300   RSET NOMBRE1$=A1$:RSET NOMBRE2$=A2$
1310   RSET NOMBRE3$=A3$:RSET NOMBRE4$=A4$
1320 '
1330 ' Impresión de los datos alineados a la derecha
1340 '
1350 PRINT NOMBRE1$: PRINT NOMBRE2$: PRINT NOMBRE3$: PRINT NOMBRE4$
1360 '
1370 END
```

```
PEDRO
ANA
LUISA
JORGE

PEDRO
ANA
LUISA
JORGE

                PEDRO
                  ANA
                LUISA
                JORGE
```

asigna a la cadena A$ una longitud de 10 caracteres y la inicializa con sólo espacios en blanco. El valor N ha de ser un número entero. El Basic aloja las cadenas de forma dinámica. La longitud de una variable de cadena (número de caracteres que contiene) puede, por tanto, variar a lo largo del programa. En fase de impresión y de memorización en disco, se plantean problemas de alineación, que cabe resolver con las instrucciones SPACE$ y LSET (o RSET). En la pág. 383 se muestra el listado de un programa que lee en vídeo cuatro nombres (A1$, etc., instrucciones de 1090 a 1120)*.

La primera fase de impresión tiene por objeto los datos tal como son introducidos (instrucción 1220); la segunda tiene lugar con alineación a la izquierda y la última con alineación a la derecha (las instrucciones de impresión están contenidas en las líneas 1260 y 1350).

En muchas ocasiones hay que crear una cadena de longitud parametrizada, por ejemplo, igual al valor del resultado de un cálculo, pero en algunas máquinas no existe la instrucción SPACE$. En estos casos hay que escribir un bucle que construya una cadena de espacios en blanco de una determinada longitud. El diagrama de un bucle de este tipo se muestra en el gráfico interior, mientras que la codificación de las instrucciones se describirá en el apartado dedicado a los bucles.

Llamando la rutina con N = 20, el resultado es el mismo que el de la instrucción A$ = SPACE$(20).

En la pág. 386 se muestra un programa de creación de una agenda. El significado de cada instrucción se explica en los comentarios. Este programa, en la forma expuesta, puede leer y escribir un solo nombre a la vez, puesto que faltan las instrucciones de bucle, que han sido sustituidas por una fila de asteriscos (líneas 150 y 410).

SPC(N)

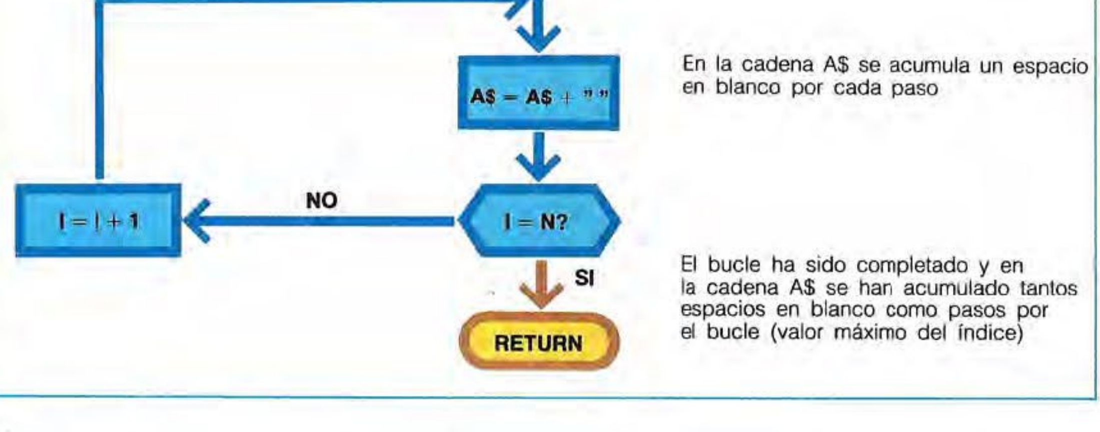
Esta instrucción (en realidad es una función, como SPACE$) no efectúa ninguna asignación, sino que tiene un efecto similar al de SPACE$; por lo tanto, se comenta en este apartado, aunque impropiamente. Esta función permite imprimir una serie de N espacios en blanco.

Por ejemplo, la instrucción PRINT SPC(6) im-

* La instrucción INPUT sirve para tomar del teclado un valor y transferirlo en la variable especificada.

GENERACION DE UNA CADENA EN BLANCO DE LONGITUD PARAMETRIZADA

Entrada: N = longitud de la cadena



TEST 11

1 / Reagrupar las siguientes constantes según su categoría (enteras, simple o doble precisión, etc.)

a: 12.57	b: "hello"	c: 79.3D4	d: 9159E6	e: &H12
f: 92735	g: &177	h: &HAF	i: "AF"	j: 2121

2 / Poniendo A% = 3, B = 3, C# = 3, D$ = "3", ¿cuáles de las siguientes expresiones son erróneas?

a: D% = A% + 7	b: D% = A% + 46721	c: D% = C# * B!
d: R! = B! * 120	e: D# = D$ + D#	f: H$ = "7" + D$

3 / Si se escribe la matriz A(10), sin más, ¿cuántos elementos la constituyen?

4 / Utilizando la función MOD, diseñar el diagrama de flujo de un programa que determine la divisibilidad por 2, 3, 5, 7 y 11 de un número cualquiera introducido mediante teclado.

5 / Generalizar el programa anterior previendo la introducción mediante teclado de los divisores (10 números máximo). El programa puede articularse en estos pasos:
1: lectura de la cantidad de números que serán introducidos (NMX);
2: bucle de lectura de los divisores (de 1 a NMX) y de memorización de los mismos en una matriz A que contenga diez elementos como máximo;
3: entrada del número del que se desea comprobar la divisibilidad (por cada uno de los divisores previamente introducidos);
4: comprobación de divisibilidad e impresión;
5: predisposición del programa para una nueva entrada.
El programa ha de finalizar si se introduce como valor de NMX el número 0.

Las soluciones, en las págs. 398 y 399.

prime seis espacios en blanco y es del todo equivalente al conjunto de las dos instrucciones A$ = SPACE$(6) y PRINT A$ aunque en este segundo caso se imprimen seis espacios en blanco, es decir, el contenido de la cadena A$. La instrucción SPC(N) se usa poco: hay instrucciones de uso más amplio que realizan las mismas funciones y otras más útiles.

SWAP

Esta instrucción, que sirve para intercambiar entre sí los valores de dos variables, puede utilizarse con cualquier tipo (entera, real o doble precisión, cadena), a condición de que las dos variables sean homogéneas. Ejemplos:

SWAP A, B — Intercambia entre sí los contenidos de A y B.
SWAP A, B# — Error. La variable A está en simple precisión, mientras que la variable B está en doble precisión.
SWAP A$, B$ — Intercambia los contenidos de las cadenas A$ y B$.

Insertando la instrucción SWAP A$, B$ en la lista de la pág. 386 antes de la impresión, por ejemplo en la línea 335, se tiene la escritura del apellido (segunda columna) en lugar del nombre y viceversa.

Bucles

En este párrafo sólo se describen las dos instrucciones principales:

FOR... NEXT...
WHILE... WEND

previstas en el Basic 80. En párrafos sucesivos se expondrán otras formas propias de versiones concretas del Basic.

FOR... NEXT... En la pág. 387, arriba, se ve el

LECTURA DE LOS DATOS E IMPRESION EN FORMA TABULAR

```
10' ** LECTURA DE LOS DATOS E IMPRESION EN FORMA TABULAR **
20'
30' DECLARACIONES:
40'
50' OPTION BASE 1 Los indices parten del valor 1
60' DEFINT I-N    Todos los nombres que empiezan con las letras de I a N
70'               son variables enteras
80' ENTRADAS:
90'
100  INPUT NOMBRE$
110  INPUT APELLIDO$
120  INPUT CALLE$
130  INPUT NUMERO$
140'                   no se tendría una impresión ordenada.
150' **********
160' La longitud de los campos debe ser:
170' Nombre    15 caracteres en la cadena A$
180' Apellido 15 caracteres en la cadena B$
190' Calle     20 caracteres en la cadena C$
200' Número     4 caracteres en la cadena D$
210'
220' Creación de las cadenas de impresión A$, B$,... etc.
230'
240 A$=SPACE$ (15)'Nombre deberá conterner NOMBRE$
250 B$=SPACE$ (15)'Apellido deberá contener APELLIDO$
260 C$=SPACE$ (20)'Calle deberá contener CALLE$
270 D$=SPACE$ (4)  'Número deberá contener NUMERO$
280'
290'  Alineado de los datos leídos en los campos de impresión
300'
310' LSET A$ = NOMBRE$
320' LSET B$ = APELLIDO$
330' LSET C$ = CALLE$
340' RSET D$ = NUMERO$ 'Las cadenas que representan valores numéricos
350'                   deben alinearse a la derecha para poderlas encolumnar.
360' Los campos de impresión A$, B$, etc. se reúnen para formar una
370' sola línea de nombre LINEA$
380'
390  LINEA$ = A$ + B$ + C$ + D$
400  PRINT LINEA$
410' **********
420' END
```

```
JOSE           PEREZ          CALLE CARDOSO       123

LUIS           SANJUAN        PLAZA VERDI        1234

ARMANDO        SETIEN         CALLE EBRO            6
```

diagrama de flujo de un bucle genérico. El índice I varía entre los valores N1 y N2 (N2>N1) con paso P. En la primera ejecución del bucle, el índice I adopta el valor N1; sucesivamente se incrementa en P y toma los valores I + P; I + 2P, etcétera, hasta llegar al valor final N2. La codificación del bucle tiene lugar con dos instrucciones: la primera especifica cuáles son los valores entre los que ha de variar el índice (N1, N2) y con qué paso; la segunda indica el punto en el cual se desea «cerrar el bucle». En este punto del programa se controla el valor del índice con respecto al límite superior (N2) y, en base al resultado, se produce la salida del bucle o la reentrada (con incremento del índice) para un nuevo paso. La instrucción que especifica los parámetros es (con referencia a la simbología del gráfico superior de la página contigua):

FOR I = N1 TO N2 STEP P

El código operativo FOR le indica a la máquina que la instrucción es un bucle; la indicación I = N1 TO N2 STEP P especifica los valores que tomará el índice. La instrucción de cierre es NEXT I (NEXT = próximo). Por lo tanto, la codifi-

ESQUEMA LOGICO DE UN BUCLE

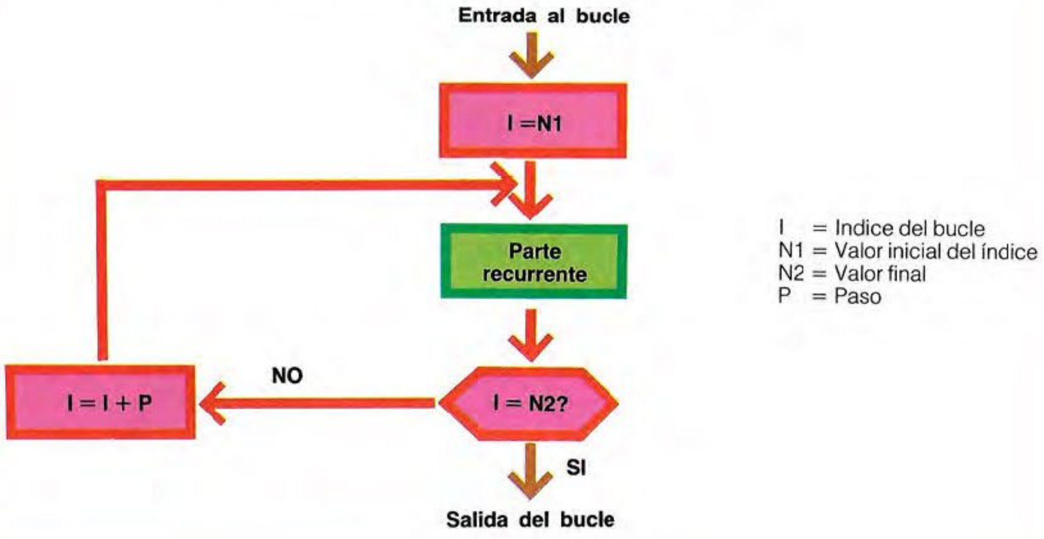
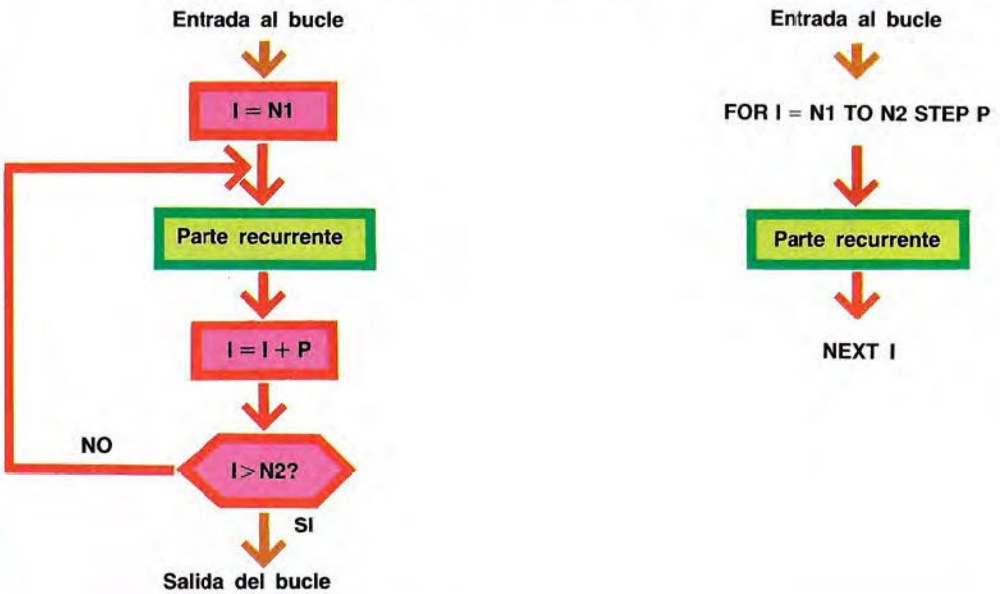


DIAGRAMA DE DESARROLLO DE UN BUCLE



cación completa del diagrama de la pág. 387 (arriba) es:

```
20 FOR I = N1 TO N2 STEP P
(Parte recurrente)
1270 NEXT I
```

Dicho diagrama es el equivalente lógico del programa codificado, pero no su forma exacta. El mecanismo de ejecución real se muestra también en la pág. 387, en el gráfico inferior.

La diferencia está en la posición del cálculo del nuevo valor de I (I = I + P). A nivel lógico, este cálculo ha de efectuarse sólo si la condición I = N2 no se ha dado todavía (diagrama superior), mientras que si el índice I ha alcanzado el valor N2, el bucle ha de ser interrumpido sin calcular un nuevo valor de I. De esta forma, en salida del bucle se tendría I = N2. En realidad, el intérprete Basic (o el compilador) procede de forma distinta: primero incrementa el índice (I = I + P en el gráfico inferior de pág. 387) y luego comprueba si el bucle ha de ser interrumpido.

Comparando los dos diagramas, está claro que en el segundo caso no se puede interrumpir con I = N2, puesto que el último valor de I (I = N2) es calculado después de la parte recurrente, y los correspondientes cálculos no han sido realizados con I = N2. La salida del bucle ha de producirse bajo la condición I > N2. El desarrollo del bucle es transparente para el programador: todos los controles, las decisiones y los incrementos son efectuados automáticamente por el intérprete. Sin embargo, hay que puntualizar que a la salida de cada bucle el índice no tiene un valor igual al del extremo superior (N2), sino que es igual al extremo superior más un paso (N2 + P). Por ejemplo, el bucle

```
FOR N = 3 TO 11 STEP 2
(Parte recurrente)
NEXT N
```

termina con el valor N = 11 + 2 = 13 (en este caso el índice se designa con la letra N).

Los bucles pueden desarrollarse sólo entre extremos enteros y con paso entero: la instrucción FOR 1 = 2.9 TO 3.1 STEP 0.1 es, por lo tanto, errónea. En el caso de que haya que realizar un bucle con valores no enteros, antes hay que convertirlos en enteros, por ejemplo multiplicándolos por 10, para luego dividirlos por 10 en el interior del bucle, en la parte recurrente, para así recuperar la parte decimal. En el listado inferior se muestra un programa que imprime los números comprendidos entre 2.1 y 3 con paso 0.1 utilizando la técnica descrita.

El intérprete Basic asume siempre los parámetros de un bucle como enteros. Se pueden, por tanto, utilizar también nombres simbólicos no explícitamente declarados como enteros (instrucción DEFINT...); el sistema se encargará de tomar sólo la parte entera.

Aunque está admitido, el uso de variables no enteras no es conveniente. Efectivamente, ocupan más espacio de memoria y obligan al intér-

EJEMPLO DE BUCLE CON VALORES NO ENTEROS

```
100'
110' ** EJEMPLO DE BUCLE CON VALORES NO ENTEROS **
120'
130' El programa imprime los números entre 2.1 y 3 con paso 0.1:
140'   Valor inicial del índice= 2.1   se convierte en 21
150'   Valor final             = 3     se convierte en 30
160'   Paso                    = 0.1   se convierte en 10
170' Se tiene, pues, un factor multiplicador 10, y por dicho valor
180' debe dividirse el índice antes de imprimirlo
190'
200'
210' N1=2.1*10:N2=3*10:P=.1*10      'Parámetros multiplicados por 10
220' FOR I=N1 TO N2 STEP P          'BUCLE
230' R=I/10
240' LPRINT R;
250' NEXT I
260' END


2.1  2.2  2.3  2.4  2.5  2.6  2.7  2.8  2.9  3
```

EJECUCION DE UN BUCLE EN PANTALLA



La fotografía muestra el listado en pantalla de un programa que ejemplifica la utilización de un bucle FOR... NEXT... El programa pregunta al operador cuál es el valor máximo (NF) del índice del bucle y cuál es el paso (S). Obsérvese que el paso S es el incremento del índice I, mientras que en fase de visualización (o impresión) con el nombre de PASO se presenta el valor del contador K, que se incrementa en 1 para cada iteración del bucle. Por tanto, en este último caso PASO tiene el significado de número de iteración.



Esta fotografía y la siguiente muestran dos ejecuciones distintas del programa. En el primer caso, tras la pregunta "¿cuál es el valor máximo del índice?", el operador introduce el número 8 e impone en la introducción siguiente que el paso sea igual a 2. El programa se encarga de visualizar en cada incremento el valor del índice I, tanto en el bucle directo como en el inverso.



En este caso se impone NF = 8 y S = 1. Al dividir por dos el paso, manteniendo igual NF, el bucle se ejecuta el doble de veces que en el caso anterior. Al final de la ejecución, el sistema responde con el mensaje de sentencia cumplida (Ok).

EJECUCION DE UN BUCLE INTERPRETADO Y COMPILADO

```
10' ** TABLA COMPARATIVA DE LOS TIEMPOS DE EJECUCION DE
20'     UN BUCLE CON INDICE REAL Y ENTERO
30' OPTION BASE 1 'Esta instrucción sirve para hacer
40'                  que los índices partan del valor 1
50  INPUT "N CON R=REAL ",N 'Esta instrucción se
60'                              explicará más adelante
70  FOR R=1 TO N STEP 2
80  NEXT R
90  PRINT N
100  DEFINT R          'DEFINE "R" ENTERO
110  DEFINT N
120  INPUT "N CON R=ENTERO"; N 'Esta instrucción se
130'                              explicará más adelante
140  FOR R=1 TO N STEP 2
150  NEXT R
160  PRINT N
170  END
```

N.° de vueltas del bucle	Programa interpretado (tiempos en segundos)		Programa compilado (tiempos en segundos)	
	R = Real	R = Entero	R = Real	R = Entero
10'000	5	4	3.5	0.5
15'000	8	6	5	1
25'000	14	9.5	9	1
32'000	17.5	12	12	1

prete a realizar funciones innecesarias (en este caso concreto, la conversión en enteros) con la consiguiente pérdida de tiempo en la realización del bucle. Recuérdese que el Basic interpretado es muy lento en la ejecución, y cada complicación del trabajo repercute notablemente en el tiempo de realización de los programas. Para dar una idea de los órdenes de magnitud, en la tabla superior se enumeran, en segundos, los tiempos necesarios para la ejecución de un simple bucle (ver listado) sin cálculos para algunos valores de los límites. Como se ve, hay una notable diferencia entre el tiempo de ejecución con variables enteras o reales y, sobre todo, entre el Basic interpretado y el compilado.

En los bucles, el valor del paso, si no se declara, se considera igual a 1. Así, las dos instrucciones

FOR I = N1 TO N2 STEP 1
FOR I = N1 TO N2

son del todo equivalentes. Los valores de los extremos (N1, N2) pueden ser calculados en la misma instrucción de bucle; por ejemplo

FOR I = 5 + N TO 20 + N

es una forma válida de parametrizar los límites. Los valores de los extremos dependen en este caso del valor que se dé a N antes de entrar en el bucle. Una forma particular de cálculo de los extremos del bucle se presenta cuando uno de los dos se calcula utilizando el valor del índice. Por ejemplo, con las instrucciones

10 I = 2
20 FOR I = 6 TO I + 25

el límite superior (I + 25) se obtiene a partir del valor del índice dado previamente (instrucción 10 I = 2). En general, el intérprete Basic prepara antes el valor del límite superior y luego el del límite inferior. La instrucción 10 asigna el valor I

= 2, por lo que en la 20 (I + 25) el límite superior es 2 + 25 = 27; por lo tanto, el bucle se desarrolla entre los valores I = 6 (I = 6 aparece en la línea 20 tras el cálculo I + 25) e I = 27.
Algunos intérpretes, sin embargo, preparan primero el valor inicial y luego el final; en este caso, la instrucción 10 no tendría peso alguno, puesto que en la instrucción 20 se pone primero I = 6 y luego se calcula el otro extremo I + 25 = 31, y el bucle se desarrolla entre 6 y 31.
Esta metodología es característica de los intérpretes Basic en versiones anteriores a las actuales. Los bucles pueden estar «nidificados», es decir, contenidos uno dentro de otro, con tal de que los más internos sean cerrados antes que

EJEMPLO DE TRES BUCLES NIDIFICADOS

```
100'
110' **EJEMPLO DE TRES BUCLES NIDIFICADOS **
120'
130   DEFINT I-N           'Los índices se declaran enteros
140   FOR I=1 TO 5         'Primer bucle (es el último en ser cerrado - línea 230)
145   PRINT                'Sirve para espaciar el bucle de 1 a 5
150   PRINT I
160   FOR J=3 TO 7 STEP 2   'Segundo bucle (línea 220)
170   K=I*J
180   PRINT K;
190   FOR L=10 TO 12              'Tercer bucle (línea 210)
200   PRINT L;
210   NEXT L
220   NEXT J                      ** Compárese la secuencia de los cierres con la
230   NEXT I                         correspondiente posición de inicio de bucle

RUN

 1
 3    10    11    12     5    10    11    12     7    10    11    12
 2
 6    10    11    12    10    10    11    12    14    10    11    12
 3
 9    10    11    12    15    10    11    12    21    10    11    12
 4
12    10    11    12    20    10    11    12    28    10    11    12
 5
15    10    11    12    25    10    11    12    35    10    11    12

100 DEFINT I-N
110 FOR L=3 TO 21
120 A=2*L+10
130 FOR N=7 TO 15
140 B=N*A
150 NEXT L
160 NEXT N

RUN

NEXT without FOR in 160

Los cierres de los dos bucles se invierten, generando así un error
```

SUBRUTINA DE GENERACION DE CADENAS

```
1000'
1010' ** SUBRUTINA DE GENERACION DE CADENAS **
1020'
1030' ENTRADAS
1040'            LS = Longitud en caracteres de la línea
1050'            B$ = Símbolo que ha de estar contenido en la cadena
1060'
1070' SALIDAS
1080'            A$ = Línea de la longitud deseada llenada con el
1090'                 carácter B$
1100  A$=""      'La línea está inicializada con longitud nula
1110  FOR I=1 TO LS 'BUCLE de la LONGITUD
1120  A$=A$+B$        'en A$ se acumula el símbolo B$ para un número
1130'                  de veces igual a LS
1140  NEXT I
1150  RETURN
1160
```

EJEMPLO DE UTILIZACION DE LA RUTINA 1000

```
100 '
110 ' ** EJEMPLO DE UTILIZACION DE LA RUTINA 1000 (generación de una cadena)
120 '
130 DEFINT I-N
140 B$="*"               'La línea está llena con el carácter *
150 LS=15                'y tiene longitud 15
160 GOSUB 1000           'llama la subrutina de generación
170 LPRINT A$
180 '
190 B$="X"               'nuevo símbolo
200 GOSUB 1000           'en la llamada no hay asignado un nuevo valor de LS
210 LPRINT A$            'por tanto la longitud sigue siendo 15
220 '
230 B$="1"
240 LS=5
250 GOSUB 1000           'En esta llamada se varían tanto el simbolo como la
260 LPRINT A$             'longitud
270 '
280 LS=280
290 GOSUB 1000           'Esta llamada contiene un error: LS=280 no puede
300 LPRINT A$            'ser aceptado, las líneas contienen máx. 255 carac--
310 '                     teres
320 END
```

los más externos (ver el capítulo de los diagramas de flujo). En la pág. 391, arriba, se dan dos ejemplos: el primero es correcto, el segundo contiene un error en el cierre de un bucle.

En la pág. 384 se ve el diagrama de flujo para la generación de una cadena de longitud parametrizada que contiene sólo espacios en blanco. La traducción en Basic de este diagrama es un caso de utilización de los bucles. Lo único que falta añadir es la parametrización.

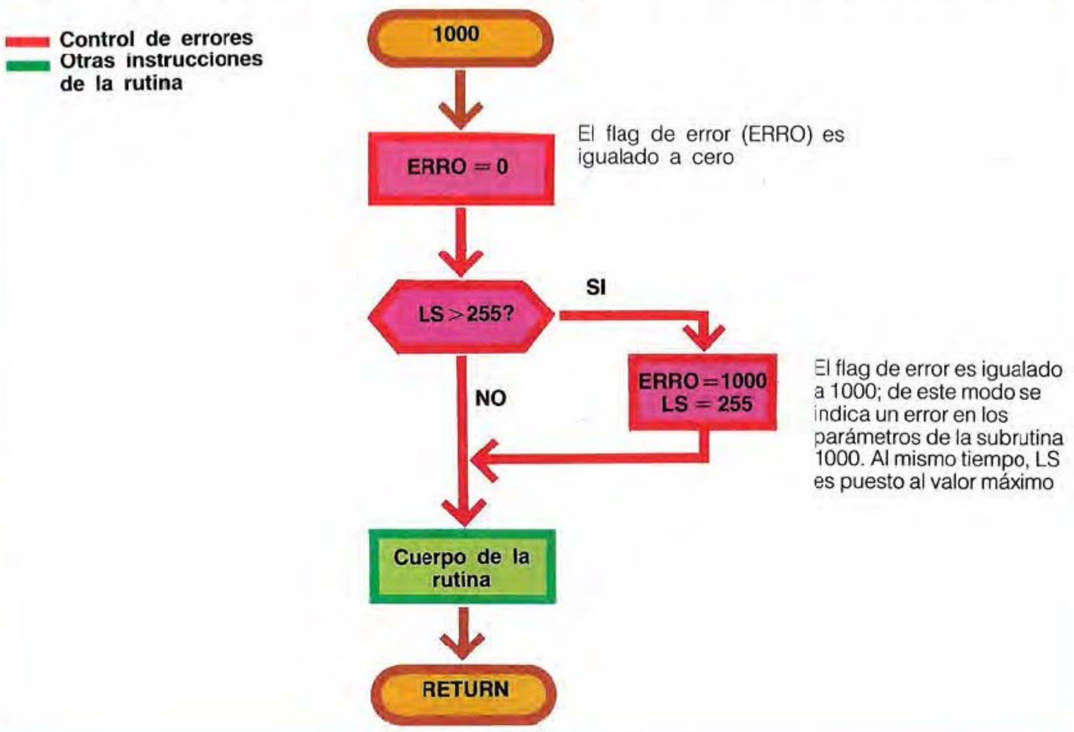
Además de dar como parámetro la longitud de la cadena, se da también el único símbolo que se desea que aparezca en la misma; de este modo es posible generar cadenas de longitud deseada y que contengan un símbolo cualquiera. En el primer listado de arriba se muestra la subrutina, mientras que en el segundo se ilustra un ejemplo de utilización de la misma. En el ejemplo se incluye una llamada con el parámetro del valor equivocado (LS>255); si en la rutina no hay controles, se produce un error y el programa se detiene. La mejor solución consiste en disponer en la entrada de la subrutina un control de validación de los parámetros transmitidos por el programa que llama. En caso de error, los parámetros se igualan a un valor con-

vencional y se activa un flag de error. Así no se detiene el programa, y el flag de error, transmitido al programa que llama, puede utilizarse para determinar qué rutina ha sido llamada con valores erróneos y las correcciones a efectuar. En el gráfico inferior y en el de la pág. 394 se ven las modificaciones al diagrama de flujo de la subrutina 1000 y del programa principal para incluir el control del parámetro LS.

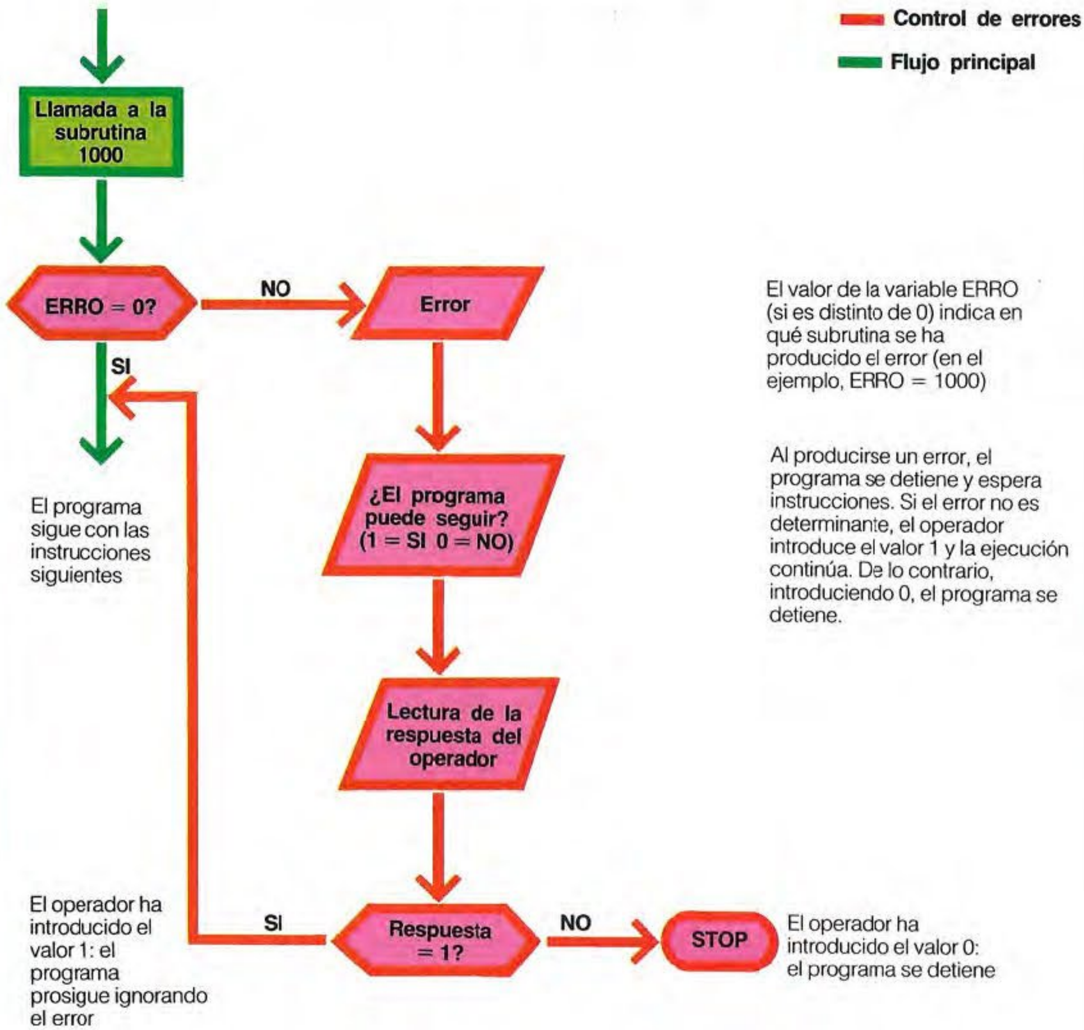
WHILE... WEND. La secuencia de las instrucciones WHILE... WEND activa otra forma de bucle. Puede considerarse como un bucle cuya ejecución está condicionada al hecho de que una determinada condición (el resultado de un cálculo, el valor de un flag, etc.) sea verdadera. Una condición es «verdadera» cuando su valor es distinto de cero, y «falsa» cuando es igual a cero. La condición de ser verdadera se indica normalmente con la palabra «true», y tiene el significado de no-cero; la condición opuesta es «falsa», con el significado de cero.
La forma de esta instrucción es:

```
10 WHILE expresión
(Parte recurrente)
120 WEND
```

EJEMPLO DE UTILIZACION DE LA SUBRUTINA DE GENERACION DE CADENAS



EJEMPLO DE LLAMADA A LA SUBRUTINA 1000 CON CONTROL DE LOS ERRORES



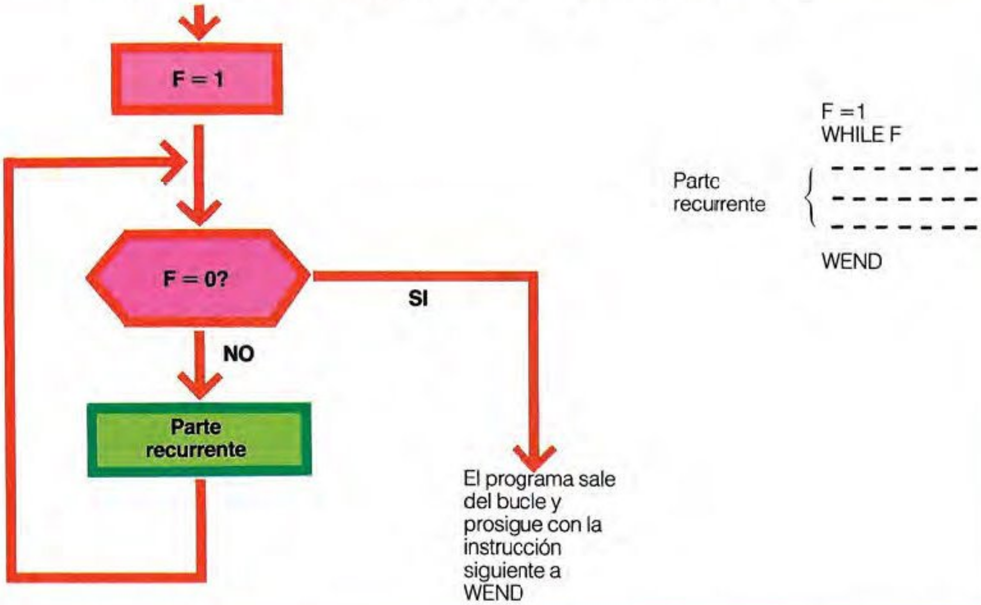
Se evalúa el cálculo (o el parámetro) indicado con el término genérico «expresión», y si tiene un valor distinto de cero se ejecutan todas las instrucciones recurrentes comprendidas entre WHILE (línea 10) y WEND (línea 120).

En ese momento el programa vuelve a la instrucción WHILE y comprueba de nuevo el valor de «expresión»; si el resultado es distinto de cero, las instrucciones recurrentes son ejecutadas de nuevo. El desarrollo del programa sigue el camino descrito hasta que el resultado de «expresión» es distinto de cero. En ese momento, el bucle es abandonado y el programa prosigue con la instrucción que sigue al código WEND.

En el gráfico se ve el diagrama del bucle, en el cual el término genérico «expresión» se indica con F. Su estructura y funcionamiento son muy distintos de los de la otra forma de bucle (FOR... NEXT...): en ésta, los límites –y por tanto el número de veces que se realiza el bucle– están determinados a priori (extremos de variación del índice), mientras que con la instrucción WHILE la parte recurrente puede realizarse indefinidamente, hasta que no se modifique la condición representada por «expresión».

Naturalmente, la parte recurrente ha de contener las oportunas instrucciones que, bajo determinadas condiciones, modifiquen el valor de «expresión», pues de lo contrario nunca se saldría del bucle. En el gráfico de la pág. 396 se

DIAGRAMA DE LAS INSTRUCCIONES WHILE... WEND

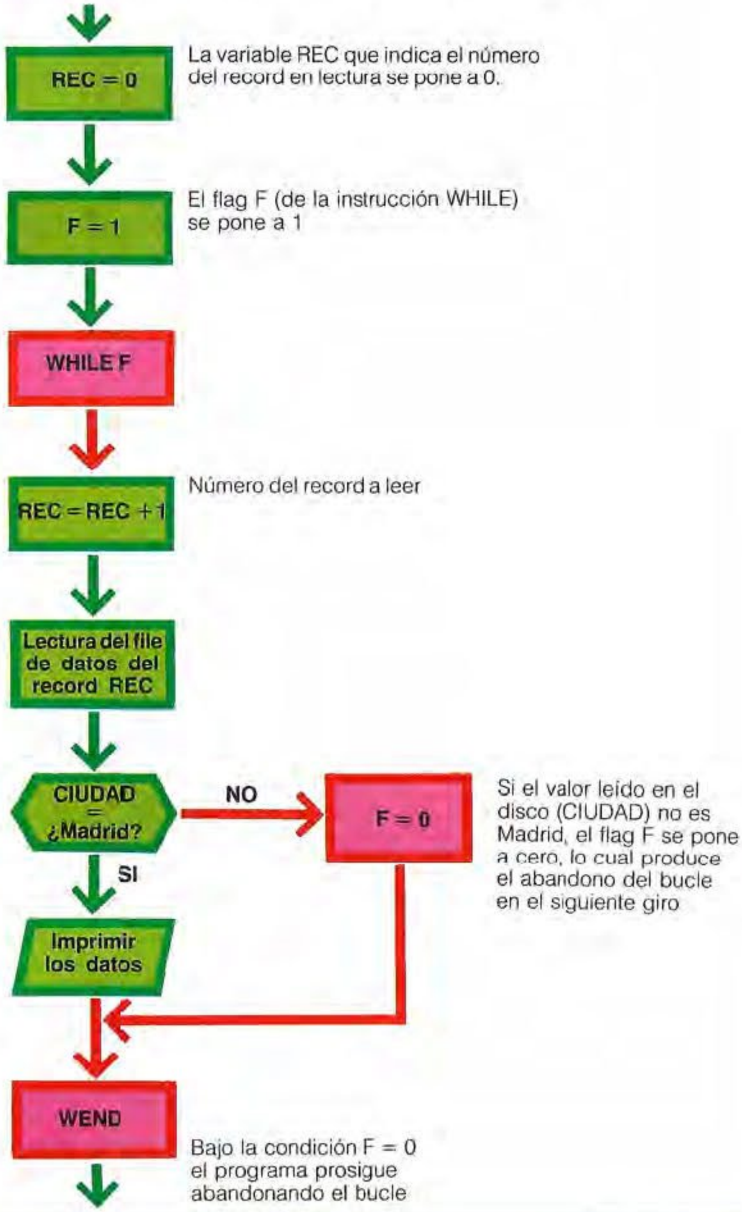


PROGRAMA QUE EJEMPLIFICA LA INSTRUCCION WHILE... WEND

```
10' ** PROGRAMA QUE EJEMPLIFICA LA INSTRUCCION WHILE...WEND **
20'
30' FILE =  EJEMPL
40'
50      F=1         INICIALIZACION DEL INDICADOR
60      WHILE F  INICIO DEL BUCLE
70  INPUT "CIUDAD";CIUDAD$
80  IF CIUDAD$<> "MADRID" THEN F=0 'LA CONDICION DE NO IGUALDAD
90                                      CIERRA EL BUCLE
100  LPRINT "CIUDAD  =" ;CIUDAD$, "CONDICION VERDADERA"
110  WEND
120  LPRINT "FIN DE PROGRAMA, CIUDAD = "CIUDAD$, "CONDICION FALSA"
130  END
```

```
     CIUDAD =MADRID CONDICION VERDADERA
     CIUDAD =MADRID CONDICION VERDADERA
     CIUDAD =MADRID CONDICION VERDADERA
     CIUDAD =MADRID CONDICION VERDADERA
     CIUDAD =VALENCIA CONDICION FALSA
     FINAL DE PROGRAMA, CIUDAD =VALENCIA          CONDICION FALSA
```

APLICACION DE LAS INSTRUCCIONES WHILE... WEND



muestra el diagrama de flujo de un ejemplo de aplicación. El programa de aplicación debe leer los records de un file de datos que contiene los apellidos y direcciones de un directorio, y debe imprimir los datos leídos mientras la ciudad (contenida en el campo de direcciones) sea Madrid. Al variar el nombre de la ciudad, el bucle de lectura debe ser abandonado, y el programa debe proseguir con otras funciones. En la pág. 395, abajo, se indica un programa que desarrolla el bucle de control del diagrama de flujo de esta página. En él se han omitido todas las instrucciones de gestión de disco, y los datos se han entrado por teclado (instrucción 70). El programa no tiene utilidad práctica.

Instrucciones de salto

A este grupo pertenecen las instrucciones que permiten «saltar» de un punto a otro del programa. Este tipo de instrucciones no debe confun-

dirse con las llamadas a las subrutinas, aunque pueden parecer similares. En la llamada a una subrutina, el control se transfiere a las instrucciones de la subrutina, que pueden encontrarse en un punto cualquiera del programa, pero al término de la ejecución es reemprendido por la línea que sigue a la que contiene la llamada. En realidad, no se «salta» ninguna instrucción; la ejecución de una parte del programa sólo se envía al interior de la subrutina. En cambio, en las instrucciones de salto, no se tiene un retorno automático hacia las instrucciones que se han dejado. El programa prosigue desde la línea de llegada del salto hacia los números de línea más altos, y la parte del programa que se ha saltado ya no se ejecuta, a menos que no sea reclamada expresamente por el programador (con otra instrucción de salto). El objeto de las instrucciones de salto es permitir diversos recorridos por el interior del programa.
La elección de uno u otro recorrido puede producirse según determinados resultados de los cálculos o para determinados valores suministrados a la máquina en la introducción de datos.

GOTO... Es la instrucción de salto más sencilla. La sintaxis es GOTO n, donde n representa el número de línea a la que debe saltar el programa. Así, GOTO 153 dirige el programa a la línea 153, y la parte comprendida entre la instrucción GOTO 153 y la línea 153 no se ejecuta.
La única condición que debe tenerse presente en la utilización de esta instrucción es que la línea a la que se dirige la ejecución debe existir (puede ser incluso un simple comentario).
La instrucción GOTO se llama «salto no condicionado» porque siempre se realiza. Existen instrucciones «condicionadas», que sólo se realizan si se verifican algunas condiciones.

ON... GOTO... Es la otra instrucción de salto. El comportamiento es similar al de la precedente con la posibilidad adicional de direccionar varias líneas simultáneamente según sea el valor de una variable. Por ejemplo, la forma completa de la instrucción puede ser:

ON V GOTO 100,250,300,1610,2000

Según el valor de V se realizará el salto correspondiente.
Para V = 1 se activa el salto a la instrucción 100, para V = 2 se salta a la línea 250 y así sucesivamente, hasta V = 5 que activa el salto a la instrucción 2000.
Si el valor de V es cero o mayor que el número de las líneas de llegada indicadas en la instrucción (5 en el ejemplo), el programa continúa con la instrucción que sigue inmediatamente a la del salto. Por ejemplo, si en la anterior instrucción se pone V = 0 o V = 6, no se activa ningún salto, y la ejecución del programa prosigue sin salto.
Los límites de validez del índice (V en el ejemplo) son 0 y 255; para valores negativos o mayores que 255, se tiene error y el programa se detiene. La variable utilizada como elemento de selección (V en el ejemplo) siempre debe ser entera; incluso si no se declara explícitamente así, sólo se considera la parte entera.
La instrucción ON... GOTO... también puede contener una expresión en lugar de la variable. En este caso, primero se valora la expresión y, después, tomando como indicador el resultado, se realiza el salto. Por ejemplo, en la instrucción ON K + 6 GOTO 10,12,21,70, primero se evalúa la suma K + 6 y después, según el resultado, se realiza el salto correspondiente (para K + 6 = 1 salta a 10, para K + 6 = 2 salta a 12, etc.).
En el ejemplo anterior se ha sugerido un método para adoptar valores negativos del índice. El resultado K + 6 = 1 sólo puede obtenerse si K vale –5 (–5 + 6 = 6 – 5 = 1); sumando a este valor negativo, utilizable directamente en la instrucción, una cantidad adecuada, se obtienen valores positivos (de 1 en adelante).
Por ejemplo, si quisiésemos utilizar una variable con valores comprendidos entre –5 y –10 deberíamos sumar 11, obteniendo así un parámetro que varía entre 1 (11 – 10) y 6 (11 – 5).
Obsérvese que la solución más obvia, la de cambiar simplemente el signo, no es equivalente. En el caso de variaciones entre –5 y –10, al cambiar el signo se tendría un valor entre 5 y 10, con lo que se podría indicar del quinto número de línea después del GOTO hasta el décimo. En cambio, sumando 11, se tiene un resultado comprendido entre 1 y 6, que puede direccionar desde el primer número de línea después del GOTO hasta el sexto. Las dos situaciones se comparan a continuación.

- **Primer método (suma de un valor positivo)**
  ON K + 11 GOTO 120,36,100,10,21,60
  Para K variando entre –10 y –5, la cantidad K + 11 varía entre 1 (se realiza la instrucción 120) y 6 (se realiza la instrucción 60).
- **Segundo método (cambio de signo)**
  L = –K

SOLUCIONES DEL TEST 11

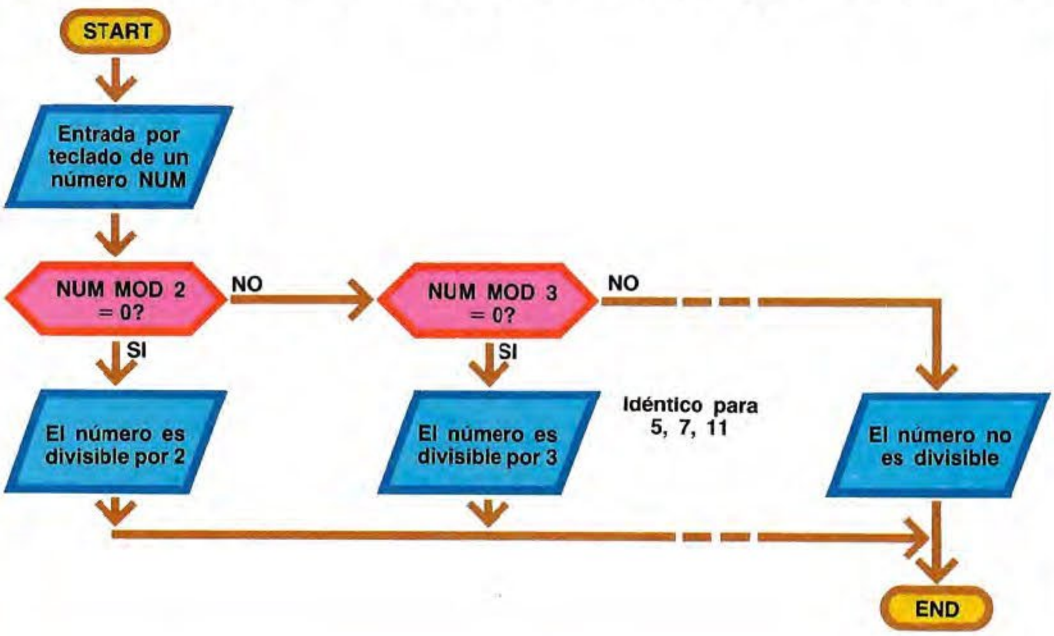
1 / a, d, f = reales en simple precisión; j = entero; b, i = constantes de cadena; c = real en doble precisión; g = octal; e, h = hexadecimales.

2 / Las expresiones erróneas son:
b: las constantes D% y A% son enteras, mientras que el número 46721 no puede serlo (es superior a 32767);
c: la constante D% es entera, mientras que C# está en doble precisión y B! es real en simple precisión (el producto da como resultado 9, por lo que en este caso concreto el cálculo se efectúa de todas formas);
e: D # es real, mientras que D$ es una cadena; no se puede pasar una cadena (que está representada en ASCII) a una variable real (que tiene una representación numérica).
La expresión H$ = "7" + D$ da como resultado H$ = "73", puesto que el símbolo +, tratándose de cadenas, tiene el significado de unión, es decir, genera una cadena formada por el conjunto de las otras.

3 / La numeración de las matrices comienza por 0. Por tanto, salvo indicación contraria, A(10) contiene 11 elementos (del elemento número 0 al 10).

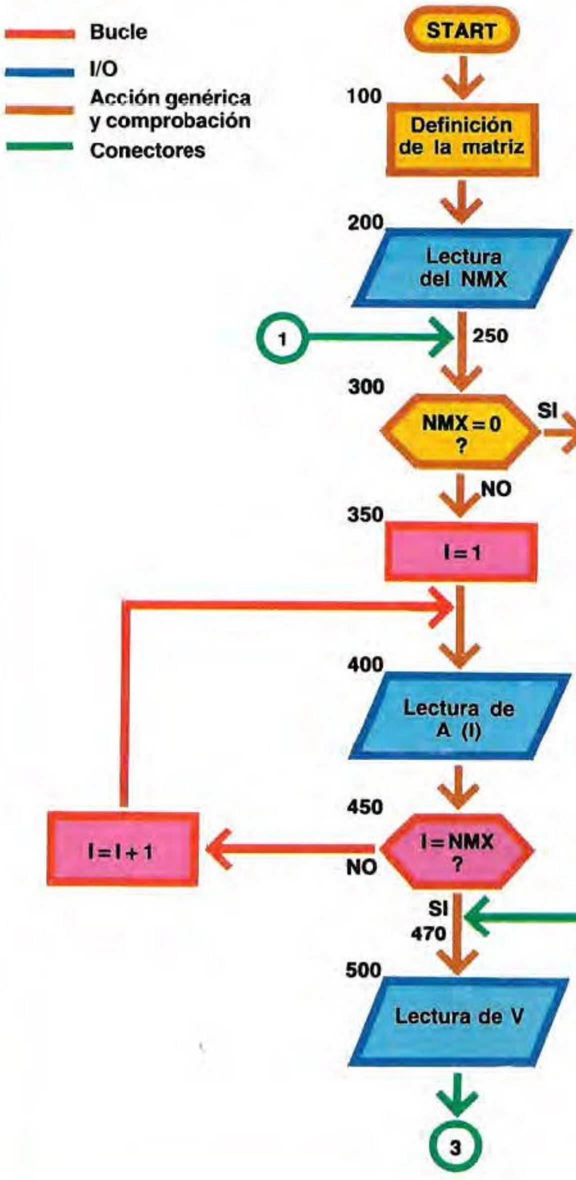
4 y **5** / Los diagramas pedidos se muestran en las figuras siguientes. El primero prevé una serie de cálculos del valor NUM MOD... (para los números 2, 3, 5, 7 y 11), cada uno seguido de una comprobación del resultado: si dicho resultado es cero, el número introducido (NUM) es divisible. Esta forma de desarrollar el programa es sólo indicativa; en el diagrama se muestra la forma que hay que usar a nivel operativo.

PROCEDIMIENTO DE CONTROL DE DIVISIBILIDAD (PRIMERA VERSION)



Con esta solución indicativa se repite cinco veces el mismo cálculo y la misma comprobación. Las funciones utilizadas (MOD y comprobación de cero) pueden parametrizarse y, por tanto, escribirse una sola vez. De esta manera, en las distintas situaciones bastará con dar los valores numéricos concretos del cálculo en curso (los parámetros).
El segundo diagrama muestra este planteamiento optimizado.

PROCEDIMIENTO DE CONTROL DE DIVISIBILIDAD (SEGUNDA VERSION)



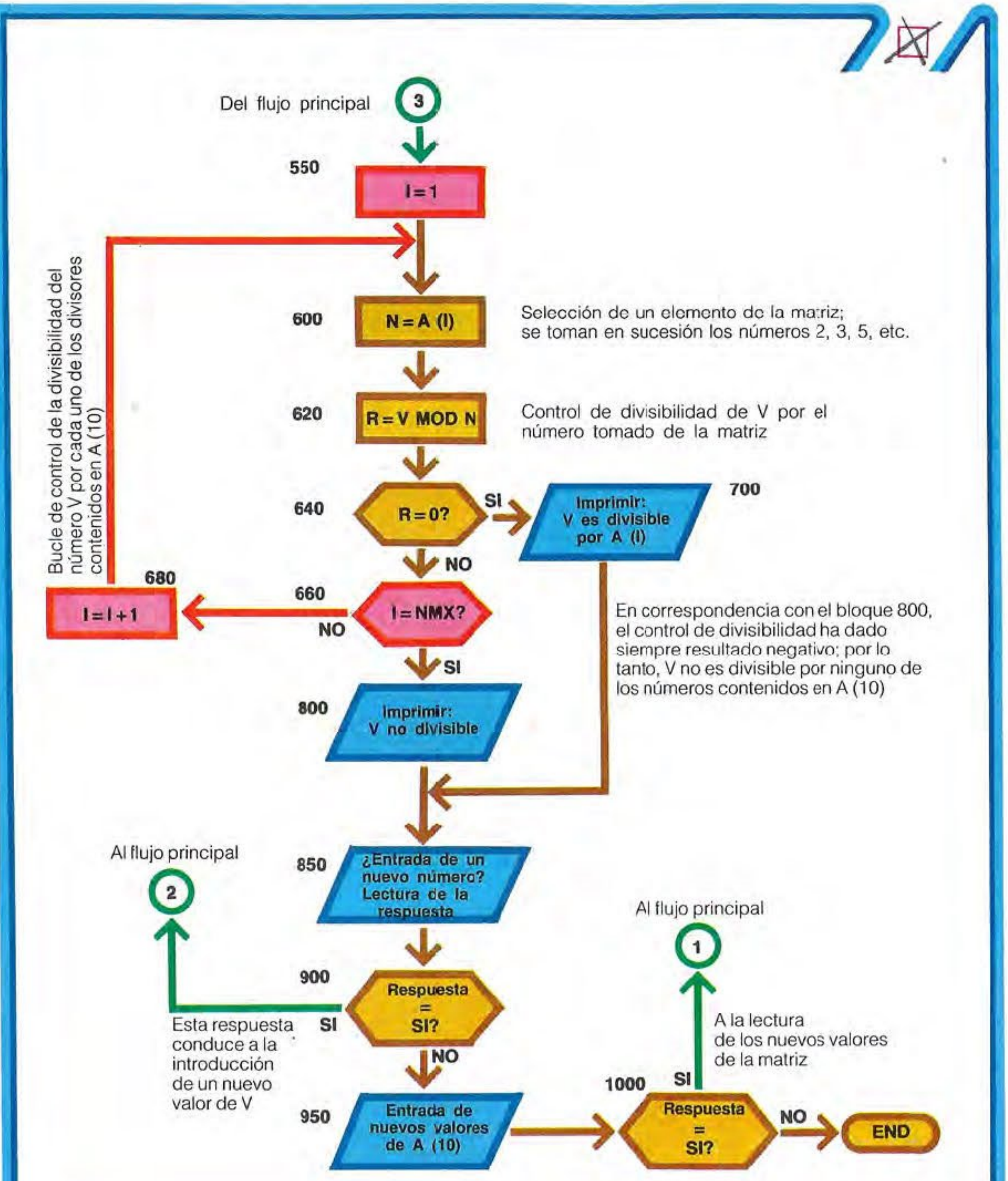
En este bloque se define la base de numeración de los elementos de la matriz (en este caso, 1) y se da la instrucción para reservar un área de memoria de diez posiciones bajo el nombre A (10). En la digitación habrá que insertar OPTION BASE 1

Lectura del número de los elementos de A (10); en el ejemplo es NMX = 5 (2, 3, 5, 7, 11)

Condición para terminar el programa

Bucle de lectura de los elementos de A (10) (divisores). En el ejemplo considerado hay que dar, de uno en uno, los valores 2, 3, 5, 7, 11

Lectura del número V del que se desea comprobar la divisibilidad



ON L GOTO ?,?,?,?,60,21,10,100,36,120
El valor de L se iguala al valor de K cambiado de signo (positivo). Como K varía entre –10 y –5, el parámetro L varía entre 10 y 5. Para conservar la correspondencia entre los valores de K y las líneas a ejecutar hay que insertar cuatro números de línea ficticios e invertir el orden de los números válidos respecto al caso anterior. Las cuatro primeras direcciones indicadas con el símbolo ? no se utilizan nunca, dado el campo de variación concreto de L (de 10 a 5).

El primer método (suma de un valor positivo) es más «limpio» y tiene menos riesgo de error. La claridad y la sencillez son requisitos fundamentales para una buena programación. Conviene utilizar alguna instrucción más de las estrictamente necesarias, con tal de evitar instrucciones demasiado complejas y poco claras.
Un ejemplo de aplicación de esta instrucción se ve en la subrutina para el cálculo de las áreas de algunas figuras planas (ver gráfico superior de pág. 373).
En el listado correspondiente se indica sólo que la selección del tipo de figura tiene lugar mediante el parámetro K. Con K = 1 se elige el cuadrado (instrucción 1660); para K = 2, el rectángulo (instrucción 1680), y así hasta el valor K = 5, al que corresponde el círculo (instrucción 1770).
La selección puede obtenerse con la sentencia:

ON K GOTO 1660,1680,1710,1740,1770

A continuación se muestra el diagrama de flujo completo y el listado correspondientes a esta nueva instrucción (se han eliminado todos los comentarios anteriores).

Las instrucciones de cálculo (1670, 1700, etc.) no se especifican, pues son idénticas a las anteriores. La línea 1656 es una «trampa» que detiene el programa si llega un valor de K no previsto (menor que 1 o mayor que 5).

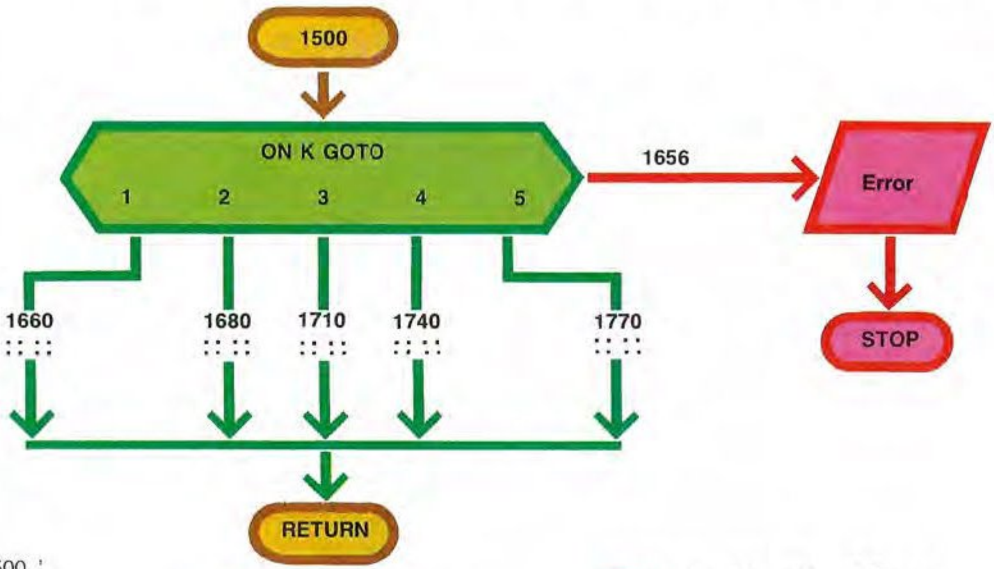
Instrucciones condicionales

Una instrucción se llama condicional cuando su ejecución depende del valor que tomen determinadas variables.
Todas las instrucciones pueden hacerse condicionales escribiéndolas en la forma siguiente:

IF condición THEN instrucción

Las palabras IF (si...) y THEN (entonces...) indican a la máquina que la instrucción es del tipo condicional; por tanto, habrá que evaluar la condición y, si se cumple, se ejecutará la instrucción que sigue al código THEN.
La condición puede estar representada por una simple variable (en este caso, la condición es cierta si la variable es distinta de cero), por un cálculo aritmético o bien por una expresión lógica. En la pág. 406 pueden observarse algunos ejemplos típicos.

DIAGRAMA Y LISTADO DE UNA INSTRUCCION ON... GOTO...




```
1500 '
1655 ON K GOTO 1660, 1680, 1710, 1740, 1770
1656 PRINT "ERROR": STOP
1660 ' CUADRADO (K = 1)
....  .....
....  .....
```

¿Cuál es el futuro del ordenador?

El desarrollo de las aplicaciones de la informática ha sido, en los últimos años, tan estimulante como para dar lugar entre los especialistas a un amplio movimiento de opinión propenso a preconizar posibilidades de evolución casi ilimitadas. Una ojeada al pasado reciente, a las perspectivas y expectativas que orientaban hace algunos años la búsqueda de nuevos horizontes, ha inducido a H.R.J. Grosch a recomendar a los entusiastas «a toda costa» que no caigan en un triunfalismo irracional. El doctor Grosch ha sido presidente de la ACM (Association for Computing Machinery) y ha ocupado numerosos cargos en la administración estatal y en algunas grandes empresas estadounidenses. Vale, pues, la pena tener en cuenta sus consideraciones acerca de esta cuestión.

La tranquila acumulación de ideas contenidas en los estantes de una biblioteca ha dejado lugar a una aventura intelectual que recuerda uno de esos sugestivos pósters de las islas Hawai: una maravillosa cascada que cae sobre bañistas que nadan en un profundo remanso. Como ocurre con el póster, la tentación es grande. La experiencia de la actual explosión del saber es impactante, por no decir peligrosa: lo mismo que nadar bajo una cascada.

El problema estriba no sólo en la profundidad del remanso, sino también en el fragor y la violencia del torrente. Las publicaciones especializadas, los medios de información en general, el entusiasmo que despiertan entre los jóvenes la ciencia ficción, los videojuegos y los ordenadores personales, en casa o en la escuela, nos llevan a aceptar y a desear conocer mejor las posibilidades de los ordenadores, las comunicaciones digitales a escala mundial, los bancos de datos, llenos de informaciones valiosísimas. He dedicado una buena parte de mi vida a promover estos instrumentos (es decir, a nadar en ese remanso maravilloso) y ha valido la pena. Soy entusiasta, pero prudente. Hace algunos años nos prometieron un lenguaje de programación independiente de la máquina y común a los grandes ordenadores. Nos prometieron la traducción del lenguaje máquina y la aparición de sistemas de gestión integrados. Nos dijeron que los programadores dejarían de existir, que los ordenadores podrían leer los caracteres impresos y jugar al ajedrez mejor que el hombre. También nos dijeron que un potente ordenador tendría el tamaño de un reloj de pulsera y podría comprender el lenguaje hablado.

Hoy nos dicen que el teclado desaparecerá pronto, que los programadores desaparecerán realmente, que los ordenadores operarán con conceptos más que con datos, que un millón de procesadores, conectados en serie, resolverán los problemas hasta ahora demasiado complejos para el hombre, que, en fin, nuevos lenguajes especialmente concebidos enseñarán a los niños más fácil y rápidamente que los viejos métodos.

¿Cuántas de estas promesas se han cumplido? Y por lo que respecta a las futuras, ¿cuánto habrá que esperar para que un nuevo hardware, tan potente, el software y la capacidad de elaboración llegue a las oficinas, las fábricas y las escuelas?

Lo primero que hay que tener en cuenta es que el éxito de los años cincuenta, sesenta y setenta ha llegado mucho más lentamente de lo que se esperaba. Es cierto que para elaborar los datos comerciales existe el Cobol, completamente estandarizado e independiente de la unidad central de elaboración, pero hoy, veinte años después de su aparición, hasta los programas en Cobol más simples exigen aún algunas puestas a punto cuando se transfieren, por ejemplo, al IBM, al Burroughs o al ICL. Los mecanismos de entrada/salida y la complejidad del software lo exigen. Existen los mini-Cobol y micro-Cobol, pero no están en absoluto estandarizados. La transposición hacia abajo sería útil, pero es imposible; hacia arriba, sería de desear, pero resulta aún extremadamente difícil.

Los intentos de traducción de los lenguajes y el control mediante el idioma inglés hablado, aunque muy simplificado, no han tenido éxito. Hace treinta años eran un bello sueño. Hace veinte años numerosos equipos trabajaban con ahínco en universidades y empresas: yo mismo he dirigido una. Pero el problema era demasiado arduo y los resultados hubieran sido antieconómicos. Incluso la realización de un «diccionario electrónico», que tan útil hubiera sido para el control de la ortografía en los «procesadores de textos», resultó entonces demasiado lenta y costosa. Aunque conscientes de estas dificultades, los investigadores no querían renunciar. Finalmente, el Pentágono decidió cortar los fondos, y el proyecto terminó.

Los SIG (Sistemas Integrados de Gestión) eran, hace diez o quince años, un concepto de moda.



C. O'Rear/West Light-Grazia Neri

Las posibilidades de utilización del ordenador son prácticamente ilimitadas, pero requieren memorias de capacidades igualmente ilimitadas.

Los directivos se sentarían entre los esplendores de la informática, frente a mágicas visualizaciones, y los datos de control y las simulaciones de las futuras tendencias alternativas desfilarían a toda velocidad ante sus ojos. Esta previsión se reveló absurda, aun antes de que se hablara de ella en los artículos de fondo del Business Week. El entusiasmo aportó dinero y prestigio a la idea del banco de datos. Pero el tejido conjuntivo existente entre los datos financieros y las operaciones variaba según las circunstancias y los períodos, las decisiones relativas al emplazamiento de una fábrica dependían en parte de los problemas familiares del presidente, y las entradas del mercado exterior dependían también de los acontecimientos políticos y religiosos.

Es cierto que los ordenadores juegan muy bien al ajedrez y que han acabado con el juego de damas; pero juegan como han establecido sus programadores, aunque de forma más atenta, exhaustiva y veloz. No han inventado una nueva manera de jugar al ajedrez: sus jugadas son las mismas que han estudiado los hombres. Ciertamente, la unidad central del mayor ordenador de los años cincuenta cabe en un chip del tamaño de una escama de pescado; no obstante los discos y las unidades de cinta magnética ocupan todavía una buena parte de una sala de máquinas.

A veces, cuando el éxito está al alcance de la mano, ya no sirve. La lectura óptica de caracteres, que depende casi por completo del perfeccionamiento del hardware, es posible. Se puede leer prácticamente cualquier documento bien impreso y mecanografiado con claridad. Pero en la actualidad, las máquinas elaboradoras de textos tienen ya tantos datos, que la capacidad de adquirirlos nuevamente ya no tiene en sí gran importancia.

En mi lista de promesas para el futuro he dicho que los más entusiastas habían previsto la inminente desaparición de los programadores, un eufemismo para decir que las esperanzas de los primeros expertos en lenguajes se han visto defraudadas. Hoy día hay miles de programadores, y su número seguirá aumentando. A menudo se les designa con otros nombres: proyectistas de microplacas, expertos en informática, operadores de instrucciones numéricas. Se han



A. Nogues/Sygma-Grazia Neri

El ordenador puede eliminar desde ahora desagradables situaciones de este tipo.

difundido por todas partes y han dado vida a una nutrida progenie de software.

Cualquier ordenador dedicado a la gestión, tanto si es mini como personal, depende potencialmente de un programador, que es tan osado como para adentrarse en los meandros del software. Hay un abismo entre la forma de trabajar de un ordenador y la de un ser humano. Este abismo es superado mil veces al día por programadores capaces, por programadores mediocres y por el personal encargado del funcionamiento, del mantenimiento y también de la adquisición de datos. Podemos conseguir que sobre este abismo se tiendan puentes más anchos y menos peligrosos, situados en lugares estratégicos. Estos éxitos me entusiasman; pero la diferencia entre la lógica fría e inflexible de los 0 y 1 y la de los seres humanos, falibles y lentos pero llenos de calor o imaginación, permanece. ¿Qué puede hacer hoy en día un ordenador? Y, más concretamente, ¿qué puede ofrecer que sea nuevo y valioso, al margen de la contabilidad y la ingeniería? Mi primera reacción es preguntar: ¿qué es lo que no funciona en la contabilidad y la ingeniería? En todo el mundo, en Nueva York como en Caracas, en París como en el Punjab, hay aplicaciones prácticas y rentables en espera de ser realizadas en sectores que ya conocemos muy bien. Tendremos que reflexionar. Esas aplicaciones requieren personal que se ocupe del análisis de los programas, expertos en software, equipos encargados del funcionamiento y el mantenimiento. Hemos de mejorar en todo el mundo la selección, formación y gestión de estos organismos.

Naturalmente, necesitamos un hardware más rápido, con más capacidad, más fiable y menos costoso. Lo tendremos antes de lo que esperamos y antes de estar en condiciones de asimilarlo. Podríamos eliminar este problema si no fuera por una razón muy simple: la elección. Los inventores, productores y vendedores dirán: «¡Comprad el mío!», y hay centenares de posibilidades y miles de combinaciones. Hay, sin embargo, un detalle que los veteranos no ignoran y que las víctimas de hoy deberían conocer: casi todo funciona. Se puede perder algo de dinero o elegir una empresa en quiebra, pero, en general, el ordenador funcionará. Por otra parte, su costo representa una pequeña parte –o despreciable, pero pequeña– del costo total a presupuestar para resolver un problema de cierta envergadura. La fase de elección del problema correcto y poder contar con personal adecuado son puntos mucho más difíciles e importantes que el desafío Honeywell-IBM.

Hoy podemos servirnos de un instrumento que tiene una importancia capital y del que carecíamos hace tan sólo diez años: el microprocesador. Tanto si se trata de una máquina independiente, como de un terminal inteligente o un control de comunicaciones, el procesador de placa única (y, cada vez con más frecuencia, de microplaca única) puede hacer maravillas. Cuando se utiliza para una tarea sencilla, es decir, cuando no requiere más que uno o dos programas para funcionar, entonces su rendimiento es excelente. Pero sucede, cada vez más a menudo, que la aparición de un sistema coincida con la de otros, y el problema de la compatibilidad y la estandarización se vuelve enorme. En consecuencia, mi punto de vista difiere del más difundido: conviene elegir, si es posible, un gran sistema central, gestionado de manera muy profesional y que determine costos bien definidos. Si esto no es posible, es mejor conservar el tipo de gestión, los estándares y la forma de programar típicos del centro, aunque

se descentralice el hardware. Sólo como última alternativa hay que permitir que cada función siga su propio curso.

Los profesores y los jóvenes emprendedores afirman que dentro de unos meses –los más honrados dicen dentro de unos años– todo será barrido por el reconocimiento de la voz y el control mediante el lenguaje hablado. Habrá estructuras nuevas y fantásticas, el arseniuro de galio sustituirá al silicio, y los sistemas ya no necesitarán ir acompañados de las instrucciones para su uso. La razón por la que he insistido tanto en el pasado es porque, aunque en número reducido y con escasa experiencia, ya escuchamos estas historias hace diez años. Ciertamente, algunas se han vuelto realidad: se utilizará el galio y las uniones Josephson, pero todo ello tendrá escasa trascendencia.

Algunos objetivos, como las interfaces completas para el lenguaje hablado, son literalmente imposibles de alcanzar. Otros, como las redes generales totalmente transparentes, aunque parezcan demasiado ambiciosos, son abordables, pero habrán de pasar decenios antes de que se pueda pensar en aplicaciones prácticas. Otros objetivos no tendrán una importancia tan grande: el trabajo con lenguajes como el Logo o el Prolog es fascinante, pero para multiplicar no prevén más que una instrucción banal. Por otra parte, poco importa el grado de asequibilidad de un sistema: siempre será más fácil de conocer con un buen manual de instrucciones.

La clave para comprender las posibilidades del ordenador era, en general, el ordenador mismo, poco difundido, falible y costoso. La gente lo miraba con admiración y competía para acercarse a él. Hoy la clave de las posibilidades del ordenador es el hombre. Por todas partes hay máquinas, y aunque los expertos son muy importantes, necesitamos sobre todo analistas y programadores. El país más pequeño y menos evolucionado entre los que están en vías de desarrollo necesita centenares de estas personas; los países más desarrollados precisan miles o decenas de miles. Países altamente desarrollados como Japón o Estados Unidos podrán necesitar millones antes del fin de esta década. Las universidades no podrán de ninguna manera preparar a tanta gente. En la actualidad, no parece que las escuelas secundarias puedan lograrlo, y es dudoso que las universidades puedan formar los profesores suficientes para que las escuelas lo consigan. Esto es aún más evidente en países como Singapur, donde la voluntad, el dinero y el espíritu de adaptación social son suficientes, pero donde el volumen de trabajo es tal que resulta descorazonador.

La solución más inmediata es la utilización del ordenador para preparar, perfeccionar y profesionalizar a los analistas, programadores, ingenieros, operadores encargados del mantenimiento y cuadros. Disponemos del hardware, del software y de los sistemas didácticos necesarios. La transposición de estos últimos a lenguas distintas de la inglesa está en marcha; en cuanto a los costos son, si no bajos, al menos definidos y estables.

Yo mismo he asistido a cursos introductorios sobre el Plato y he podido observar que, mediante un curso avanzado, un experto puede familiarizarse rápidamente con un sistema de difícil asimilación. A nivel académico, he visto funcionar un importante sistema Plato centralizado en una gran universidad cerca de Ciudad del Cabo.

Las posibilidades del microprocesador son, por otra parte, de gran ayuda. Allí donde los costos, la energía y las exigencias ambientales excluyen el empleo de una importante unidad central de elaboración, actualmente se pueden instalar máquinas de oficina con un sistema didáctico capaz de enseñar el Basic o el Fortran, lectura y ortografía, aritmética y gramática, álgebra y geometría, contabilidad e ingeniería. Estas máquinas todavía no tienen acceso a los enormes bancos de datos de que podría servirse la enseñanza en general (y más concretamente la universitaria), y me consta que hay también obstáculos culturales. Pese a todo, los progresos en materia de hardware resolverán el primer problema, y quiero suponer que a medida que aumenten las técnicas de informática y los tipos de problemas tratados por ordenador, se atenuarán al menos algunas de las tensiones de índole cultural.

Resumiendo, las posibilidades actuales de los ordenadores son enormes.

No necesitamos esperar a los procesadores de información o las uniones Josephson. Los costos son accesibles y siguen disminuyendo. Pero existe un factor limitador: las personas –las personas adecuadas– y es el ordenador lo que nos ayuda a formarlas.

(Extraído de «Le pouvoir de l'ordinateur de nos jours», de H.R.J. Grosch. AGORA, n.º 5, 1983.)

IF V < > O THEN PRINT "No cero"
si V no es cero, imprimir la frase "No cero"
IF A > B THEN C = A + B
el cálculo C = A + B se efectúa tan sólo si A > B
IF A = B THEN 1020
si A = B el programa salta a la instrucción 1020

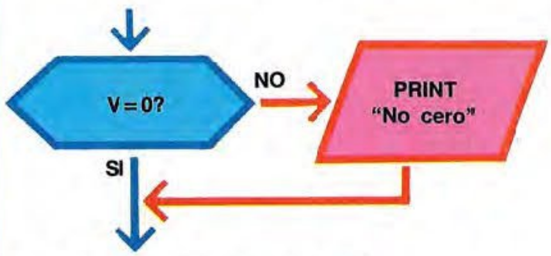
Esta última sentencia es un ejemplo de salto condicional (ver instrucción GOTO...) y puede escribirse también en la forma: IF A = B GOTO 1020. Obsérvese que la instrucción GOTO es la única que puede escribirse en forma condicional (IF... GOTO) sin el vocablo THEN.

Abajo se muestran los diagramas de las tres instrucciones anteriores y de una forma más compleja. Este último diagrama prevé (línea 10) que hayan de cumplirse dos condiciones simultáneas: A = B y (AND) C > D; en este caso, el cálculo a efectuar es F = 3 * B; en caso contrario, el cálculo es F = 5 * B.

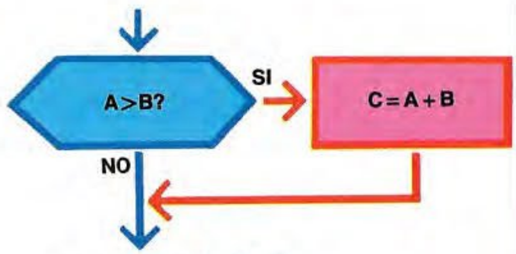
En la instrucción 10 hay que prever un salto a la instrucción que sigue a la 20; sin este salto se tendría siempre la realización del cálculo F = 5 * B. Efectivamente, en la línea 10, si las dos condiciones son verdaderas, se efectúa la otra forma de cálculo, pero inmediatamente después (si faltara GOTO 126) se ejecutaría la instrucción F = 5 * B, que superpondría el nuevo resultado al anterior. Para evitar esto, hay que prever el salto de la línea 20 cuando la comprobación (IF...) dé resultado positivo, aislando el cálculo F = 5 * B.

Las instrucciones condicionales se utilizan a menudo para acelerar la realización de un bucle. Un bucle puede servir para buscar un determinado valor en un conjunto de datos. La lógica a adoptar en este caso consiste en prepa-

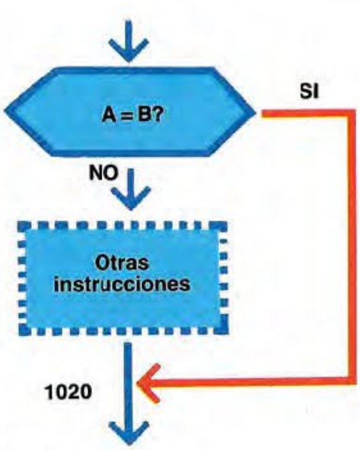
DIAGRAMAS DE ALGUNAS INSTRUCCIONES CONDICIONALES



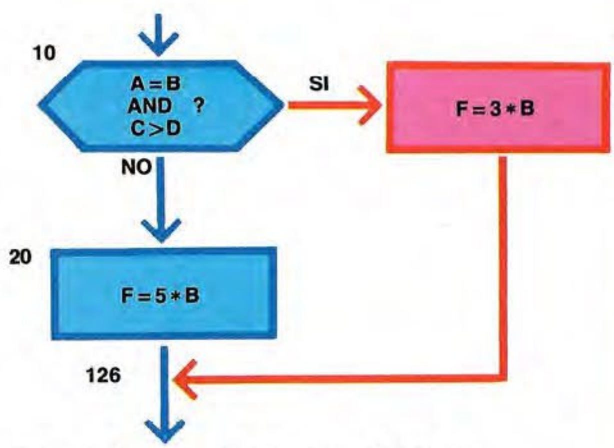
IF V < > 0 THEN PRINT "No cero"
(El símbolo < > significa distinto de...)



IF A>B THEN C=A+B

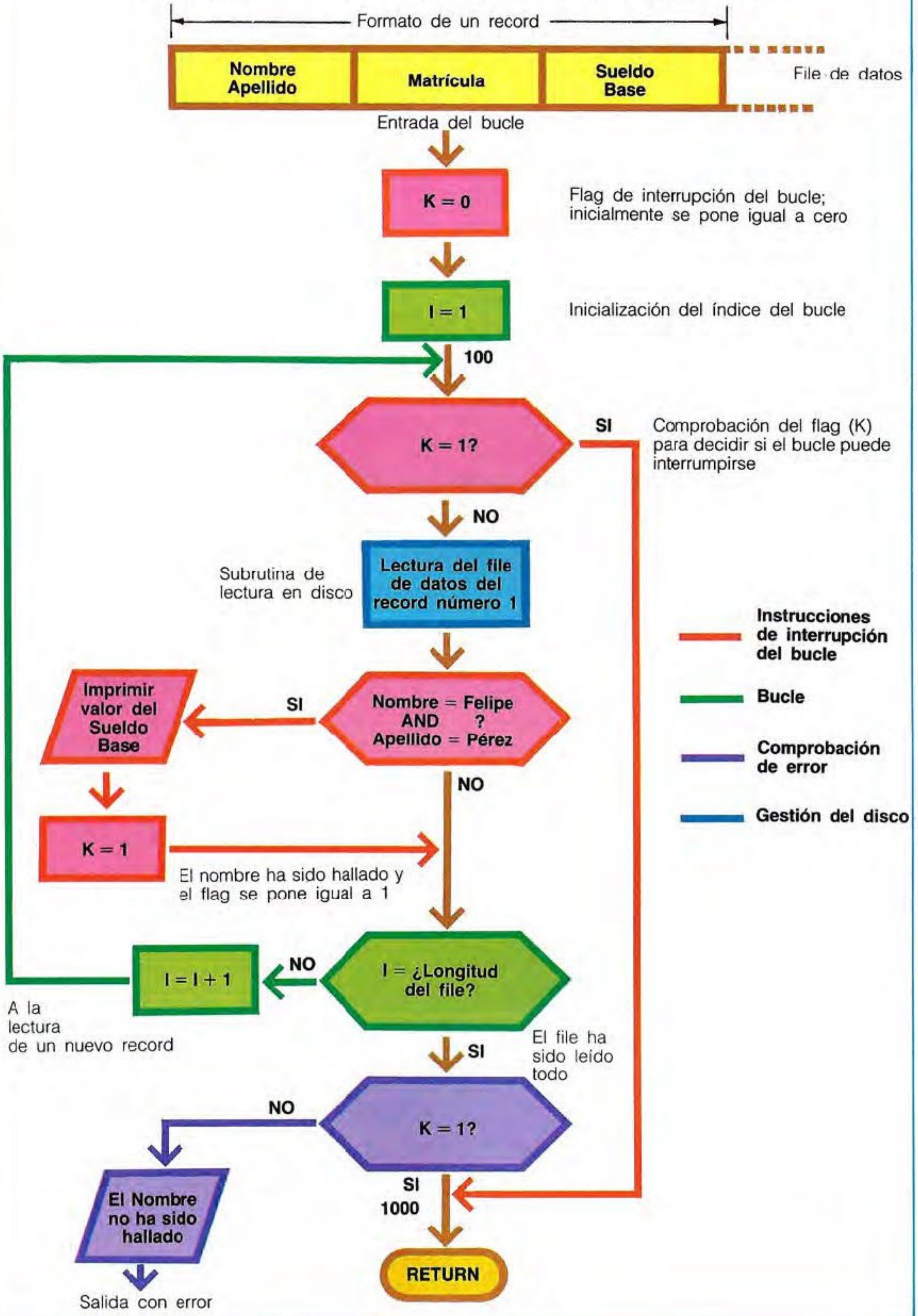


IF A=B THEN 1020
o también
IF A=B GOTO 1020



10 IF A=B AND C>D THEN F=3*B: GOTO 126
20 F=5*B
126 ' A este punto se llega desde la 10

DIAGRAMA DE BUCLE CON FLAG DE INTERRUPCION



rar un bucle que extraiga, de uno en uno, todos los datos presentes (partiendo del primero) y que los compare con el valor deseado; si la comparación es positiva, el dato ha sido hallado y el bucle puede ser interrumpido. La interrupción puede tener lugar poniendo un flag igual a 1 cuando la condición de búsqueda se cumple; dicho flag indicará a la máquina que el bucle puede ser abandonado.
En la pág. 407 se muestra un ejemplo para la búsqueda en una nómina de empleados.
El file contiene el nombre y apellido, la matrícula y el sueldo base de cada empleado, y se desea conocer el sueldo del señor Felipe Pérez. Inicialmente, el flag (de nombre K en el ejemplo) se pone igual a 0, lo que significa que la búsqueda aún no ha concluido. Al entrar en el bucle (instrucción 100) se efectúa un control del valor de K: si dicho valor es 1, el bucle es abandonado.
En las instrucciones internas del bucle, si el control del nombre da resultado positivo, es decir, el nombre es el que se busca, además de imprimir los datos se pone el flag igual a 1. Así, en la siguiente iteración del bucle, la instrucción 100 recibe K = 1 y se activa el salto al final (instrucción 1000). En otras palabras, el valor K = 1 indica que la búsqueda ha tenido éxito (en la vuelta anterior) y el programa puede terminar.
En el primer ejemplo del gráfico de la pág. 406 se muestra la forma más simple de instrucción condicional: la comprobación de un valor y la consiguiente decisión de efectuar o no su impresión. En general, el símbolo de decisión y la correspondiente instrucción (IF...) tienen dos posibles vías de salida: la primera, si la comprobación es positiva; la segunda, si da resultado negativo. En el primer ejemplo se utiliza (con la instrucción de impresión) sólo una de las dos vías y no hacen falta más advertencias. Por el contrario, en el último ejemplo del mismo gráfico hay que recurrir a una instrucción de salto (GOTO 126 al final de la línea 10) para evitar que se ejecute la línea 20 incluso en caso de comprobación positiva en la 10. Esta estructura, aunque se usa, no es la óptima.
Hay una instrucción que puede usarse en casos análogos: la instrucción ELSE, que expresa la acción a realizar en caso de que la comprobación contenida en el símbolo de decisión no haya dado resultado positivo. En otras palabras, las dos salidas del símbolo de decisión corresponden, en este caso, una a la comprobación positiva y otra a la negativa.

Las respectivas instrucciones son:

IF	(condición a cumplir)
THEN	(operaciones a efectuar si la condición es verdadera)
ELSE	(operaciones a efectuar si la condición es falsa)

Con esta nueva instrucción, la codificación del diagrama de la pág. 406 se convierte en:

```
10 IF A = B AND C > D THEN F = 3 * B
15 'Acción a realizar si la condición se cumple

20 ELSE F = 5 * B
25 'Acción a realizar en caso contrario
```

Mediante símbolos, las dos instrucciones pueden representarse del siguiente modo:

DIAGRAMA DE LAS INSTRUCCIONES IF... THEN... ELSE...



Las instrucciones IF... THEN... ELSE pueden usarse a continuación una de otra, en la misma línea. Por ejemplo, la instrucción

IF A > B THEN C = 1 ELSE C = 0

es válida, mientras que la siguiente:

IF A > B THEN C = 1 ELSE IF A = B THEN C = 0 ELSE C = –1

(que puede escribirse en una sola línea) analiza todas las situaciones que pueden presentarse en una comparación entre las variables A y B. Para interpretar esta última sentencia basta con

leer los códigos* de la siguiente forma:

IF A>B THEN C = 1
SI A mayor que B ENTONCES (poner) C = 1
ELSE IF A = B THEN C = 0
DE LO CONTRARIO SI A = B (poner) C = 0
ELSE C = – 1
DE LO CONTRARIO (poner) C = – 1

* Las sentencias, en inglés, pueden traducirse así:
IF = SI THEN = ENTONCES ELSE = DE LO CONTRARIO

Uso de las variables reales en las sentencias condicionales. La representación estándar de una variable real (es decir, que tiene una parte decimal) es en coma flotante, con un determinado número de cifras que depende de la máquina. Si el valor de la variable es el resultado de un cálculo, puede ser redondeado, por lo que tiene una imprecisión que, aunque pequeña, puede dar lugar a resultados negativos en una comprobación. Supongamos que queremos comprobar si el resultado de una operación es cero. Llamando V a la variable que contiene dicho resultado, la sentencia podría ser: IF V = 0

Realización por ordenador de una vista digitalizada de la isla de Manhattan (Nueva York).



Pierce/Wheeler Pictures-Grazia Neri



Pierce/Wheeler Pictures-Grazia Neri

TEST 12

1 / Tras las definiciones DEFINT I-N y DEFDBL Z, ¿cuáles de las siguientes asignaciones son erróneas?

a) IND = 3.14   b) R = 3.5   c) Z = P + 7.5   d) CIRC = I * ZA

2 / ¿Cuáles son los valores asignados a las variables A, B, C, D, E, F, G con las siguientes instrucciones?

```
10 READ A,B,C
20 RESTORE
30 READ D
40 RESTORE 80
50 READ E,F,G
60 DATA 5,7,21,40,60
70 DATA 3,9,10
80 DATA 15,20,75
```

3 / El siguiente programa contiene un error; ¿cuál?

```
10 X = A + B
20 READ C,D
30 RESTORE
40 READ E,F,G
50 DATA 3,7
```

4 / ¿Qué modificación hay que efectuar en el programa de la pregunta 2 para asignar a las variables E, F y G los valores 7, 21 y 40 (contenidos en la línea 60)?

5 / Este programa contiene un error que puede eliminarse borrando una línea; ¿cuál?

```
10 DEFINT I-N
20 DEFDBL Z
30 READ A,B,C
40 X = C * (A + B)
50 RESTORE
60 READ A$
70 PRINT X
80 DATA 5,3,2
90 DATA "Esto es una prueba"
```

Las soluciones, en la pág. 413.

THEN... De esta forma (que sería exacta si V fuese entero) se pueden dar errores causados por la imprecisión de V; por ejemplo, si el valor de V es 0.001, a efectos prácticos se puede asimilar al valor 0, mientras que para la máquina existe una diferencia. La prueba IF V = 0 daría resultado negativo.

Para eliminar esta posibilidad de error siempre conviene indicar, en la instrucción condicional, la precisión con la cual se desea efectuar el control. En el ejemplo anterior, si se escribe IF V<0.001 THEN..., la condición se cumple para todos los valores de V inferiores a 0.001. Este número es la precisión con la que se efectúa la comprobación. Pero tampoco en esta forma la instrucción es exacta. La variable V podría tomar valores negativos muy alejados de cero, pero menores que 0.001 (– 1000 es menor que 0.001) y la prueba daría resultados erróneos. Para estar seguros de la respuesta, hay que

considerar en la prueba el «valor absoluto» de V, es decir, el valor independientemente del signo. De este modo, un eventual valor negativo (al quitar el signo –) se vuelve positivo, y la prueba da resultados más fiables.
Para extraer el valor absoluto de una variable hay una instrucción (función) especial que veremos más adelante. Como alternativa, se puede utilizar el método ilustrado en el gráfico inferior, que consiste en convertir siempre el valor de V en un número positivo.

Llamada a la subrutina y encadenamiento de programas

La instrucción para «llamar» una subrutina es GOSUB n (siendo n el número de la línea en que comienza la subrutina). También esta instrucción puede ser condicional. Por ejemplo:

GOSUB 1250
La subrutina 1250 es llamada sin condición alguna
IF A>B THEN GOSUB 1250
En este caso el control pasa a la 1250 sólo si A>B
ON V GOSUB 1250, 720, 500
La instrucción transfiere el control a una de las subrutinas 1250, 720 o 500, en función del valor de V (ver sentencia ON... GOTO...).

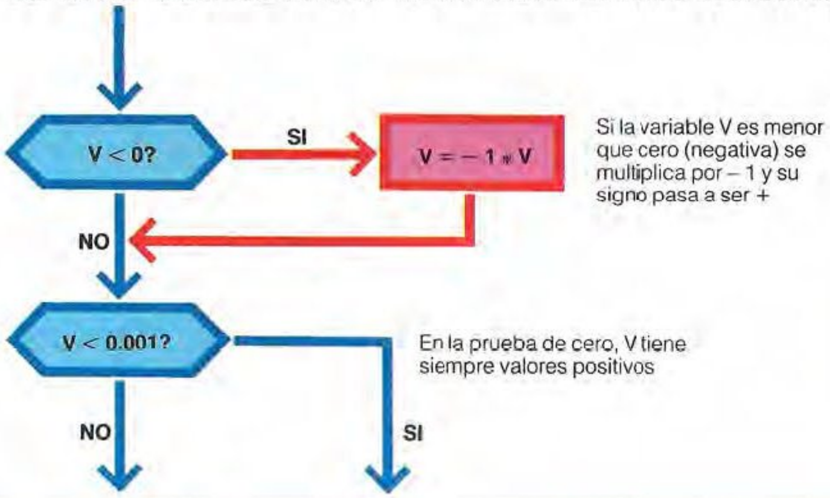
La sentencia GOSUB... es similar a la GOTO...; difiere en que el punto de llegada del salto es el inicio de una subrutina, al final de la cual la sentencia RETURN devolverá la ejecución del programa a la línea siguiente a la GOSUB...

Las subrutinas constituyen una parte integrante del programa que las utiliza y, por tanto, en fase de ejecución se cargan todas en memoria. Su uso, aunque recomendable para una buena estructuración de los programas, no siempre permite ahorrar espacio de memoria. En muchos casos, un programa no puede ser contenido enteramente en la memoria del ordenador y, por tanto, hay que subdividirlo en partes más pequeñas residentes en disco, que se cargan en memoria sólo cuando se usan (ver sistemas operativos). Cada una de estas partes es un programa en sí misma, y al ser cargada recubre las instrucciones preexistentes, que tienen la misma numeración que las que la componen.
La sentencia que permite efectuar la carga de un programa como parte de otro es CHAIN "NOMBRE", donde NOMBRE es el nombre del file en que está memorizado (en disco) el programa que hay que cargar. Por ejemplo:

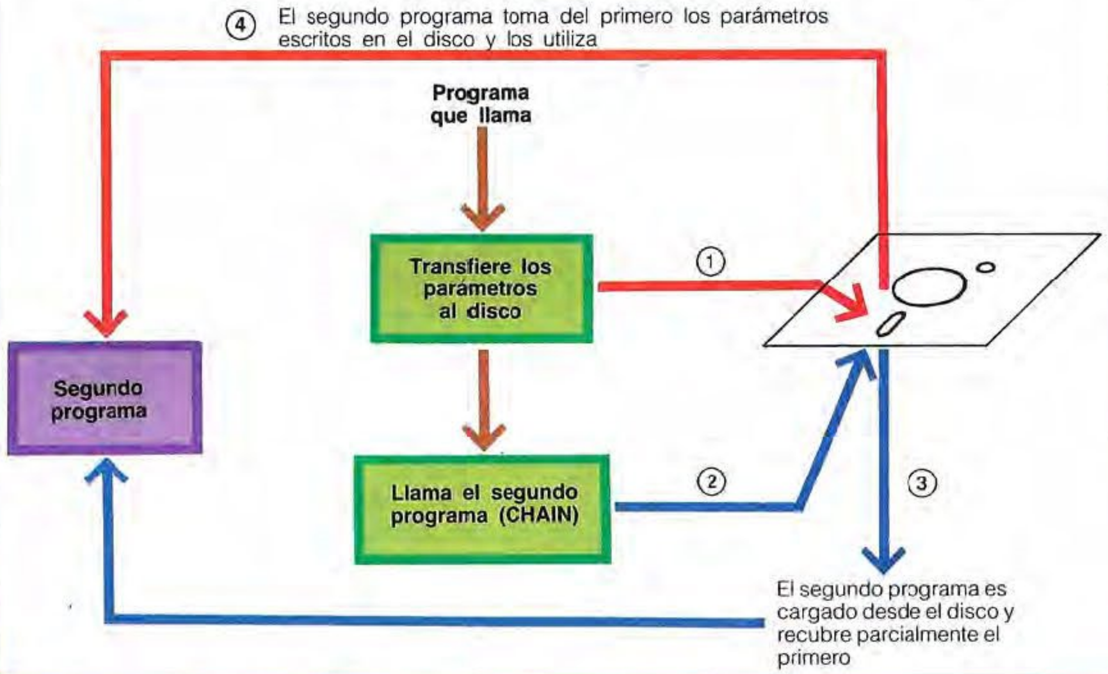
CHAIN "PRUEBA"
carga el programa que reside en el file PRUEBA
CHAIN "B:XY"
carga el programa XY desde el disco B

La sentencia CHAIN puede indicar también el número de línea desde el que ha de empezar la ejecución del programa cargado. Si se escribe CHAIN "PRUEBA",275, el programa PRUEBA es puesto en ejecución a partir de la instrucción 275. Si se omite el número (como en los ejemplos anteriores), la ejecución comienza por la primera instrucción del nuevo programa.

PRUEBA DE CERO CON EL VALOR ABSOLUTO DE UNA VARIABLE



ESQUEMA DE ENCADENAMIENTO CON PASAJE DE PARAMETROS



El programa que llama y el llamado pueden tener algunas variables a utilizar en común, en cuyo caso hay que transferir los valores del uno al otro. La transferencia puede efectuarse de tres maneras distintas:

- utilizando un file de tránsito como apoyo de los valores
- con la opción ALL
- utilizando la sentencia COMMON

En el primer caso, el programa que llama escribe en disco, en el file pertinente, todos los valores que han de ser transferidos al programa llamado. Éste, en cuanto es cargado, lee dichos valores en el file y puede utilizarlos.

En el gráfico superior se muestra el esquema lógico de esta técnica. El método es laborioso, pero a veces resulta insustituible, puesto que no todos los sistemas operativos admiten los otros dos métodos.

El segundo método consiste en especificar la opción ALL en la sentencia CHAIN. Esta opción permite transferir todos los valores de las variables de un programa a otro. Por ejemplo:

```
CHAIN "NOMBRE",152, ALL
```

tiene el siguiente significado: cargar el programa NOMBRE y ejecutarlo a partir de la instrucción 152, conservando todos los valores de las variables. Esta opción no puede usarse siempre, ya que en algunos compiladores no está prevista. En general, cabe utilizarla en el Basic interpretado, pero no en el compilado, y puesto que la forma final de los programas está en compilado, conviene no usarla.

La tercera posibilidad es la más común y también la más versátil. Puede usarse tanto en Basic interpretado como compilado, pero está incluida sólo en los sistemas operativos más evolucionados. El código COMMON declara «comunes» todas las variables indicadas; por ejemplo, COMMON A,X,N declara de uso común las variables A, X y N. Para ellas se reserva un área de memoria concreta que no queda cubierta al cargar otros programas.

La sentencia CHAIN tiene otras dos opciones: MERGE y DELETE. La opción MERGE permite cargar el nuevo programa como un «overlay», es decir, como un segmento del programa que llama, lo que permite mantener vigentes en el programa llamado todas las definiciones de clase de las variables (por ejemplo, DEFINT..., etc.), que de lo contrario habría que repetir.

La opción DELETE permite borrar el segmento cargado en overlay antes de llamar otro nuevo.

Para utilizar la opción MERGE, el programa llamado ha de estar en formato ASCII. Hay que señalar, además, que normalmente estas opciones no están previstas en los compiladores.
Resumiendo, una forma completa de la sentencia de encadenamiento es:

CHAIN MERGE "NOMBRE",1350,ALL,DELETE 185-2730

con el siguiente significado:
cargar en forma overlay el programa NOMBRE, iniciando la ejecución en la línea 1350, con transferencia completa de las variables; borrar las instrucciones entre la 185 y la 2730.
Las subrutinas externas no pertenecientes al programa principal pueden estar escritas en lenguajes distintos del Basic (Fortran, Cobol, Assembler); en este caso hacen falta instrucciones especiales para pasar los parámetros y llamar los segmentos. Estas instrucciones especiales serán descritas más adelante, en el capítulo dedicado a las técnicas de programación con lenguajes mixtos.

Ejemplo aplicativo: ¿diesel o gasolina?

En la elección del tipo de automóvil a comprar, de diesel o de gasolina, hay distintas opiniones. Normalmente se establece un mínimo de kilómetros anuales, por debajo de los cuales el diesel ya no es conveniente. El valor de dicho mínimo depende de factores como:

- diferencia entre los costos de compra de los dos tipos de vehículo
- diferencia entre los impuestos de circulación
- gastos de combustible
- valor del vehículo usado en el momento de la venta

Para que el cálculo de la conveniencia económica suministre una evaluación correcta, hay que tener en cuenta todos estos factores, en función también de la marca y del modelo concretos.
En la pág. 414 se muestra el diagrama que, en función de los parámetros del gasto (costo, im-

SOLUCIONES DEL TEST 12

1 / Las asignaciones erróneas son la a) y la d): la a), porque una variable cuyo nombre comienza por la letra I es declarada entera (DEFINT I-N); la asignación d) contiene la variable CIRC en simple precisión, mientras que ZA es declarada doble de forma implícita (DEFDBL ZA).

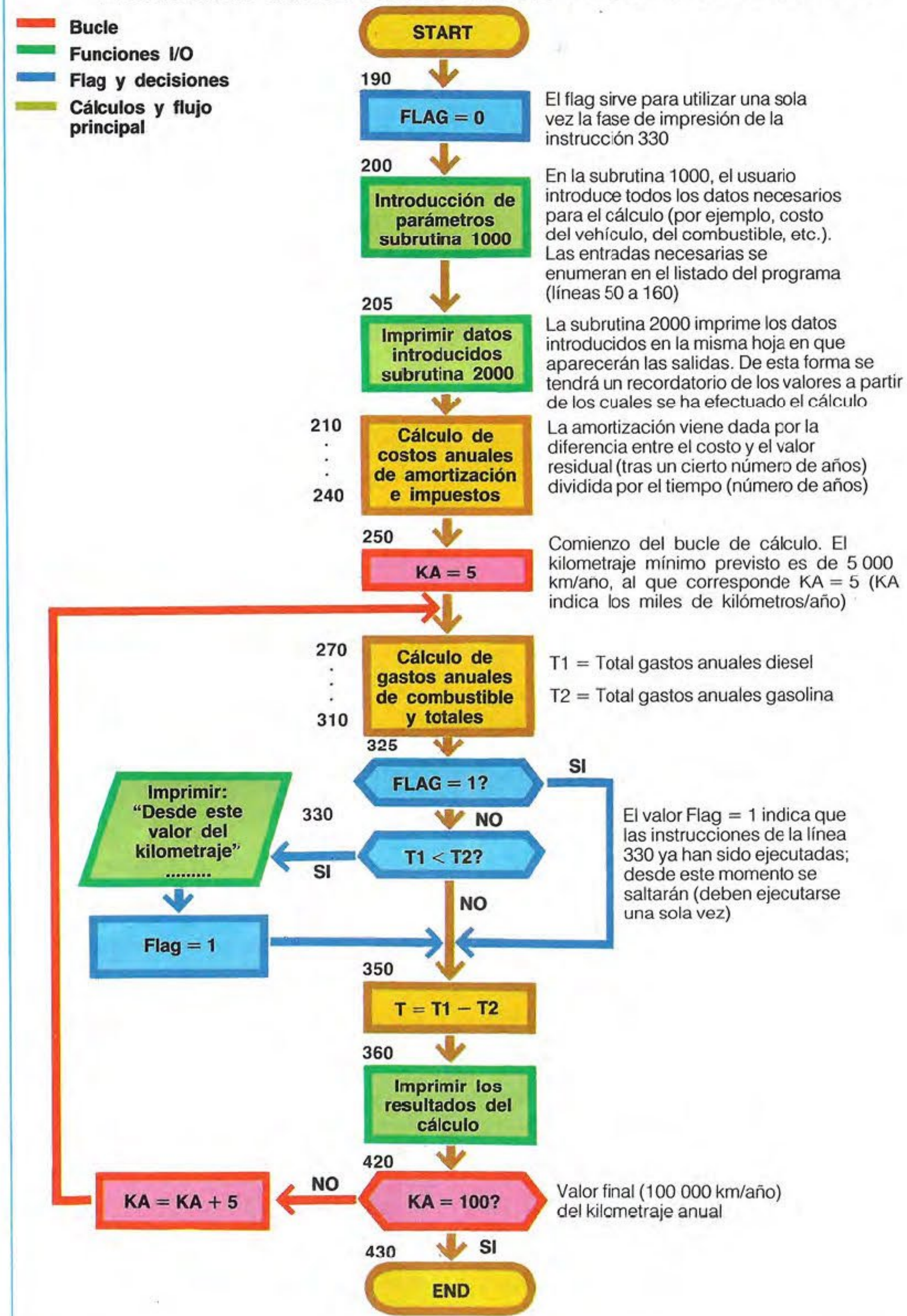
2 / La línea 10 asigna A = 5, B = 7, C = 21, y posiciona el puntero de los DATA sobre el cuarto valor de la línea 60. La sentencia 20 pone a cero este puntero y lo vuelve a situar al comienzo de DATA; por esto, a la variable D (línea 30) se le asigna el valor 5. La sentencia 40 posiciona el puntero al comienzo de DATA, que se halla en la línea 80; la siguiente lectura (línea 50) asigna E = 15, F = 20, G = 75.

3 / Tras la línea 30, el READ de la línea 40 intenta leer tres variables en un DATA que sólo contiene dos valores.

4 / Basta con quitar la línea 40. De esta manera, el puntero de los DATA permanece en el segundo valor de la línea 60 (el primer valor ha sido asignado a la variable D, línea 30). El siguiente READ (línea 50) adquiere los valores E = 7, F = 21, G = 40.

5 / La línea 50. Tras el READ de la línea 30, el puntero se posiciona al comienzo del DATA de cadena, línea 90, y puede leerse la cadena A$ (sentencia 60). La sentencia 50 lleva de nuevo el puntero al comienzo del DATA numérico (línea 80), y la línea 60 no puede ser ejecutada, pues la variable (cadena) y los datos (numéricos) son incongruentes.

DIAGRAMA DEL PROGRAMA DE CALCULO PARA LA ELECCION ECONOMICA ENTRE UN AUTOMOVIL DIESEL O DE GASOLINA



CALCULO DE CONVENIENCIA ENTRE DIESEL Y GASOLINA

```
5 ' ** CALCULO DE CONVENIENCIA ENTRE DIESEL Y GASOLINA **
6 '
7 '                FILE = GABE
8 '
10 OPTION BASE 1          'Los índices parten todos de 1
20 '
30 '
32 A$ ="££££££££.£"
35 B$ ="££££££££££"
40 ' VALORES DE ENTRADA Y VARIABLES
50 '        CG = Costo automóvil con motor diesel (gasoil)
60 '        CB = Costo automóvil con motor a gasolina (gasolina)
70 '        KG = Kilómetros recorridos con un litro de gasoil
80 '        KB = Kilómetros recorridos con un litro de gasolina
90 '        LG = Precio de un litro de gasoil (L/1)
100 '       LB = Precio de un litro de gasolina (L/1)
110 '       TG = Impuesto anual de circulación para automóvil a gasoil
120 '       TB = Impuesto anual de circulación para automóvil a gasolina
130 '       AA = Número de años previstos de utilización del vehículo
140 '       KA = Recorrido anual en miles de kilómetros
150 '       VG = Valor de recuperación del autómovil a gasoil
160 '       VB = Valor de recuperación del autómovil a gasolina
170 '
180 '
190 FLAG = 0
200 GOSUB 1000        'SUBRUTINA PARA LA ADQUISICION DE DATOS
205 GOSUB 2000        'SUBRUTINA PARA LA IMPRESION DE LOS DATOS ADQUIRIDOS
210 C1=(CG-VG)/AA     'COSTO ANUAL DE AMORTIZACION DEL AUTOMOVIL A GASOIL
220 C2=(CB-VB)/AA     'COSTO ANUAL DE AMORTIZACION DEL AUTOMOVIL A GASOLINA
230 S1=C1+TG          'COSTO ANUAL DE AMORTIZACION DEL AUTOMOVIL A GASOIL (+IMPUESTOS)
240 S2=C2+TB          'COSTO ANUAL DE AMORTIZACION DEL AUTOMOVIL A GASOLINA (+IMPUESTOS)
250 FOR KA=5 TO 100 STEP 5
260 '
270 R1=LG*KA*1000/KG   'GASTO ANUAL DE GASOIL
280 R2=LB*KA*1000/KB   'GASTO ANUAL DE GASOLINA
290 '
300 T1=S1+R1           'GASTOS ANUALES DEL AUTOMOVIL A GASOIL
310 T2=S2+R2           'GASTOS ANUALES DEL AUTOMOVIL A GASOLINA
320 '
325 IF FLAG=1 GOTO 350
330 IF T1<T2 THEN LPRINT      "A partir de este valor de kilometraje anual"
332 IF T1<T2 THEN LPRINT      "conviene el automóvil a GASOIL": FLAG=1
340 '
350 T=T1-T2
360 GOSUB 1500   'SUBRUTINA PARA LA IMPRESION DE LOS RESULTADOS
390 '
409 PRINT "INTRODUCIR UN CARACTER PARA CONTINUAR"
410 S$=INPUT$ (1)      'DETENER EL PROGRAMA Y ESPERAR LA INTRODUCCION
411 '                   DE UN CARACTER
420 NEXT KA
430 END
1000 INPUT      "COSTO AUTOMOVIL GASOIL"; CG
1010 INPUT      "COSTO AUTOMOVIL GASOLINA"; CB
1020 INPUT      "KM RECORRIDOS CON 1L DE GASOIL"; KG
1030 INPUT      "KM RECORRIDOS CON 1L DE GASOLINA"; KB
1040 INPUT      "COSTO/LITRO GASOIL"; LG
1050 INPUT      "COSTO/LITRO GASOLINA"; LB
1060 INPUT      "IMPUESTO CIRC. GASOIL"; TG
1070 INPUT      "IMPUESTO CIRC. GASOLINA"; TB
1080 INPUT      "N. DE AÑOS PREVISTOS DE UTILIZACION DEL AUTOMOVIL"; AA
1090 '
1100 INPUT      "VALOR DE RECUPERACION DEL AUTOMOVIL A GASOIL"; VG
1110 INPUT      "VALOR DE RECUPERACION DEL AUTOMOVIL A GASOLINA"; VB
1120 '

1130 RETURN
1500 ' **SUBRUTINA PARA LA IMPRESION DE LOS RESULTADOS**
1510 LPRINT "Km="; KA*1000
1520 LPRINT "GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    = ";: LPRINT USING B$;T1
1530 LPRINT "GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA = ";: LPRINT USING B$;T2
1540 LPRINT "DIFERENCIA DE GASTOS                         = ";: LPRINT USING B$;T
1550 LPRINT
1560 LPRINT
1570 RETURN
2000 '
2010 ' **SUBRUTINA PARA LA IMPRESION DE LOS DATOS ADQUIRIDOS**
2020 '
2030 C$="****CALCULO DE CONVENIENCIA DE USO ENTRE AUTOMOVIL A GASOIL Y A GASOLINA****"
2040 D$="                                 TIPO DE AUTOMOVIL"
2050 E$="DATOS DE PARTIDA"                GASOIL   GASOLINA"
2060 F$="PRECIO DE COMPRA DEL AUTO"
2070 G$="KM RECORRIDOS CON UN LITRO DE COMBUSTIBLE"
2080 H$="PRECIO DE UN LITRO DE COMBUSTIBLE (L/1)"
2090 J$="IMPUESTO ANUAL DE CIRCULACION"
2100 K$="N. DE AÑOS PREVISTOS DE UTILIZACION DEL AUTOMOVIL"
```

```
2120 M$="VALOR DE RECUPERACION DEL AÜTOMOVIL"
2130 LPRINT C$:LPRINT
2140 LPRINT D$:LPRINT
2150 LPRINT E$:LPRINT
2160 LPRINT F$;:LPRINT TAB(44):LPRINT USING B$;CG;CB:LPRINT
2170 LPRINT G$;:LPRINT TAB(44):LPRINT USING A$;KG;KB:LPRINT
2180 LPRINT H$;:LPRINT TAB(44):LPRINT USING B$;LG;LB:LPRINT
2190 LPRINT J$;:LPRINT TAB(44):LPRINT USING B$;TG;TB:LPRINT
2200 LPRINT K$;:LPRINT TAB(44):LPRINT USING B$;AA;AA:LPRINT
2220 LPRINT M$;:LPRINT TAB(44):LPRINT USING B$;VG;VB:LPRINT:LPRINT:LPRINT
2230  RETURN
```

```
**** CALCULO DE CONVENIENCIA DE USO ENTRE AUTOMOVIL A GASOIL Y A GASOLINA****

                                                     TIPO DE AUTOMOVIL

DATOS DE PARTIDA                                     GASOIL        GASOLINA

PRECIO DE LA COMPRA DEL AUTOMOVIL                    1800000       1200000

KM RECORRIDOS CON UN LITRO DE CARBURANTE                22,7          13,4

PRECIO DE UN LITRO DE CARBURANTE (L/1)                  58            93

IMPUESTO ANUAL DE CIRCULACION                         6600          4500

N. DE AÑOS PREVISTOS DE UTILIZACION DEL AUTOMOVIL       5             5

VALOR DE RECUPERACION DEL AUTOMOVIL                  600000        300000



Km = 5000
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    =        259.375
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA  =        219.201
DIFERENCIA DE GASTO                          =        40.174

Km = 10000

GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    =        272.151
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA  =        253.903
DIFERENCIA DE GASTO                          =         18.248

Para este valor de kilometraje anual es conveniente el automóvil a gasoil
KM = 15000
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    =        284.926
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA  =        288.604
DIFERENCIA DE GASTO                          =         -3.678

Km = 20000
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    =        297.701
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA  =        323.306
DIFERENCIA DE GASTO                          =        -25.605

Km = 25000
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOIL    =        310.477
GASTO ANUAL GLOBAL DEL AUTOMOVIL A GASOLINA  =        358.007
DIFERENCIA DE GASTO                          =        -47.530
```

puestos, consumos), da el costo anual total para distintos números de kilómetros, con el valor mínimo a partir del cual el motor diesel es más conveniente que el de gasolina.

En las págs. 415 y 416 se muestra el listado del programa, junto con un ejemplo de cálculo, en la hipótesis de dos automóviles de precio 1.800.000 y 1.200.000 ptas. respectivamente, con una previsión de recuperación, tras cinco años, de 600.000 y 300.000 ptas.

Como puede verse en el listado de salida (aquí arriba) la conveniencia económica está a favor del diesel a partir de los 15.000 kilómetros anuales. Este valor se obtiene a partir de las hipótesis de gastos especificadas al comienzo del cálculo, que constituyen los datos de entrada a suministrar al ordenador. Con el mismo programa se pueden variar las condiciones iniciales (datos de entrada) para obtener una nueva estimación, por ejemplo cambiando la cilindrada y, con ella, los costos y los consumos.

La decisión sobre la compra estará en función de la categoría de vehículo y del mejor compromiso entre desembolso y calidad.

Errores y su diagnóstico

Al escribir un programa es inevitable, al menos en la fase preliminar, cometer errores. Su búsqueda y eliminación plantean dificultades que aumentan al crecer el número de instrucciones; este es otro motivo por el que se procura escribir los programas subdividiéndolos en subrutinas, cada una de las cuales puede ser considerada y comprobada por separado.

Las técnicas de comprobación de los programas se describirán más adelante. En este apartado se dan las sentencias concretas para la búsqueda de errores y algunos métodos generales para su utilización. Los errores pueden dividirse en dos categorías principales:

- errores en fase de escritura del programa
- errores en fase de ejecución del programa

Los errores del primer tipo, llamados también errores de sintaxis, derivan de una formulación inexacta de las instrucciones, por ejemplo, de un error en la digitación o del hecho de no respetar el espaciado entre palabras.

Estos errores son reconocidos y evidenciados inmediatamente por el intérprete Basic.

También la corrección se puede efectuar inmediatamente, puesto que sólo se trata de reescribir la instrucción correctamente. Hay que tener en cuenta, sin embargo, que este primer diagnóstico sólo garantiza que las instrucciones son formalmente exactas, es decir, que han sido escritas usando la simbología correcta, pero no asegura el buen funcionamiento del programa en fase de ejecución.

Por ejemplo, al escribir un bucle se puede omitir, por descuido, el cierre (NEXT). A este nivel de diagnóstico, la sentencia de comienzo del bucle (FOR... TO...) es formalmente exacta y el intérprete no señala error. La falta de cierre del bucle se evidenciará más tarde, con el programa en marcha. Los errores del segundo tipo son mucho más complejos y se manifiestan sólo durante la ejecución del programa. Los métodos de búsqueda y eliminación dependen de la causa que haya generado el error.

Los principales errores en fase de ejecución pueden clasificarse así:

- errores de programación
- errores de lógica
- errores en los cáculos
- errores cometidos por el usuario al introducir los datos
- errores de sistema

Los errores de programación se deben al uso erróneo de las instrucciones. Algunos ejemplos son la omisión del cierre de un bucle o de la sentencia RETURN al final de una subrutina, la no correspondencia existente entre el número de variables que aparecen en una sentencia READ y el número de datos contenidos en la correspondiente sentencia DATA, una sentencia GOTO... que contenga un número de línea inexistente, etc.

La casuística de estos errores es muy variada y no se pueden formular métodos generales para su búsqueda y eliminación. Normalmente, el intérprete Basic incluye un buen módulo de diagnóstico, y las indicaciones que aparecen en el monitor al producirse un error de este tipo son suficientes para su identificación.

Los errores de lógica son los más complejos, ya que no generan diagnóstico; su búsqueda sólo es posible mediante un atento examen de los diagramas de flujo, o siguiendo el programa mientras desarrolla un caso ya resuelto manualmente. La comparación entre los resultados obtenidos por la máquina y los obtenidos a mano puede ayudar a identificar la zona del programa que contiene el error. El otro medio para la búsqueda de errores de lógica es el comando TRON. Al ejecutar un programa tras introducir la sentencia TRON, aparecen en pantalla todos los números de las instrucciones a medida que se van ejecutando. Así se puede seguir el camino que va recorriendo la máquina y controlar si la ejecución se desarrolla bien.

Similares a los anteriores son los errores que se cometen en la realización de los cálculos. Pueden deberse al uso incorrecto de los paréntesis o de los operadores, o a la inadecuación de la clase de variable. Se puede caer en error declarando entera (con la sentencia DEFINT...) una variable cuyo valor puede superar 32767 (o ser menor que – 32768). Sólo en este caso (error de clase) se obtiene un diagnóstico del sistema; en los demás se obtienen resultados erróneos, y tendrá que ser el programador quien busque la causa primera de los errores.

Tampoco los errores cometidos en fase de introducción de datos son denunciados, normalmente, por el sistema. Tiene que ser el progra-

ma de aplicación el que controle que los datos introducidos sean congruentes y, en caso de error, ha de emitir el diagnóstico antes de una nueva demanda. Los únicos controles operados por el sistema son los relativos al número y la clase de las variables requeridas en input.

Los errores de sistema son imputables a un mal funcionamiento de la máquina o a causas accidentales, relacionadas con las memorias masivas (por ejemplo, el operador puede olvidarse de insertar un disco, o puede colocar uno erróneo). Este tipo de error se describirá detalladamente en el capítulo dedicado a las instrucciones relativas a las unidades de disco.

Un programa bien estructurado ha de incluir al menos una subrutina para el diagnóstico y eventual corrección de errores. La llamada a esta subrutina puede efectuarse mediante la sentencia ON ERROR GOTO... (para los errores señalados por el sistema) o mediante instrucciones de diagnóstico escritas por el programador (para los errores cometidos por el operador en la fase de introducción de datos).

La sentencia ON ERROR GOTO n, donde n es el número de línea en que comienza la «trampa» para los errores, puede utilizarse una sola vez en todo el programa.

Al producirse un error cualquiera (señalado por el sistema) pueden suceder dos cosas: en el caso de que no se utilice la sentencia ON ERROR GOTO..., el sistema emite su diagnóstico y detiene la ejecución del programa. Aunque el error pueda subsanarse mediante la oportuna corrección, el programa ha salido de la fase de ejecución y habrá de volver a comenzar desde el principio. Por el contrario, si al producirse el error se utiliza la sentencia ON ERROR GOTO n, la ejecución pasa a la línea n, a partir de la cual, analizando el tipo de error, es posible decidir si se detiene la ejecución o si el operador ha de llevar a cabo una acción correctora.

El código numérico que informa sobre el tipo de error está contenido en una variable de nombre ERR, mientras que el número de la línea en la cual se ha generado el error está contenido en la variable ERL. Estos dos nombres de variables (que pueden tener siglas distintas según los tipos de Basic) están reservados al sistema y no pueden, por tanto, utilizarse para otros fines.

En la página contigua se muestra el diagrama genérico para el uso de esta instrucción. La sentencia ON ERROR GOTO 1000 insertada en el comienzo del programa direccionará a la línea 1000 en el caso de que se produzca un error. En la ramificación del punto 1000 se prevén dos errores subsanables (tipos 26 y 41). La aparición de uno de los dos determinará la llamada a la subrutina correspondiente; si el error no está comprendido entre los previstos, la sentencia ON ERROR GOTO 0 invalidará la anterior (ON ERROR GOTO 1000) y el control pasará al sistema, que, tras emitir el diagnóstico Basic normal, detendrá la ejecución.

El control de funcionalidad de las subrutinas para la corrección de errores puede realizarse con la sentencia ERROR (n), que simula la aparición del error de tipo n.

En el diagrama del ejemplo, utilizando las sentencias ERROR (26) y ERROR (41) se puede comprobar la lógica de corrección de errores.

Diagnóstico para la introducción de datos

En la fase de introducción de datos es fundamental llevar a cabo un control de validez sobre los valores suministrados al programa en el input. Las lógicas de control son dos: la primera prevé un control inmediato en la misma rutina de lectura; la segunda, la escritura de una rutina de control especial que será llamada desde todos los puntos del programa que contienen las instrucciones de introducción. El primer método se utiliza si el control no es repetitivo, sino ligado a una sola fase de introducción; el segundo, para concentrar en una subrutina todas las instrucciones de diagnóstico, en caso de que deba usarse en varios puntos del programa.

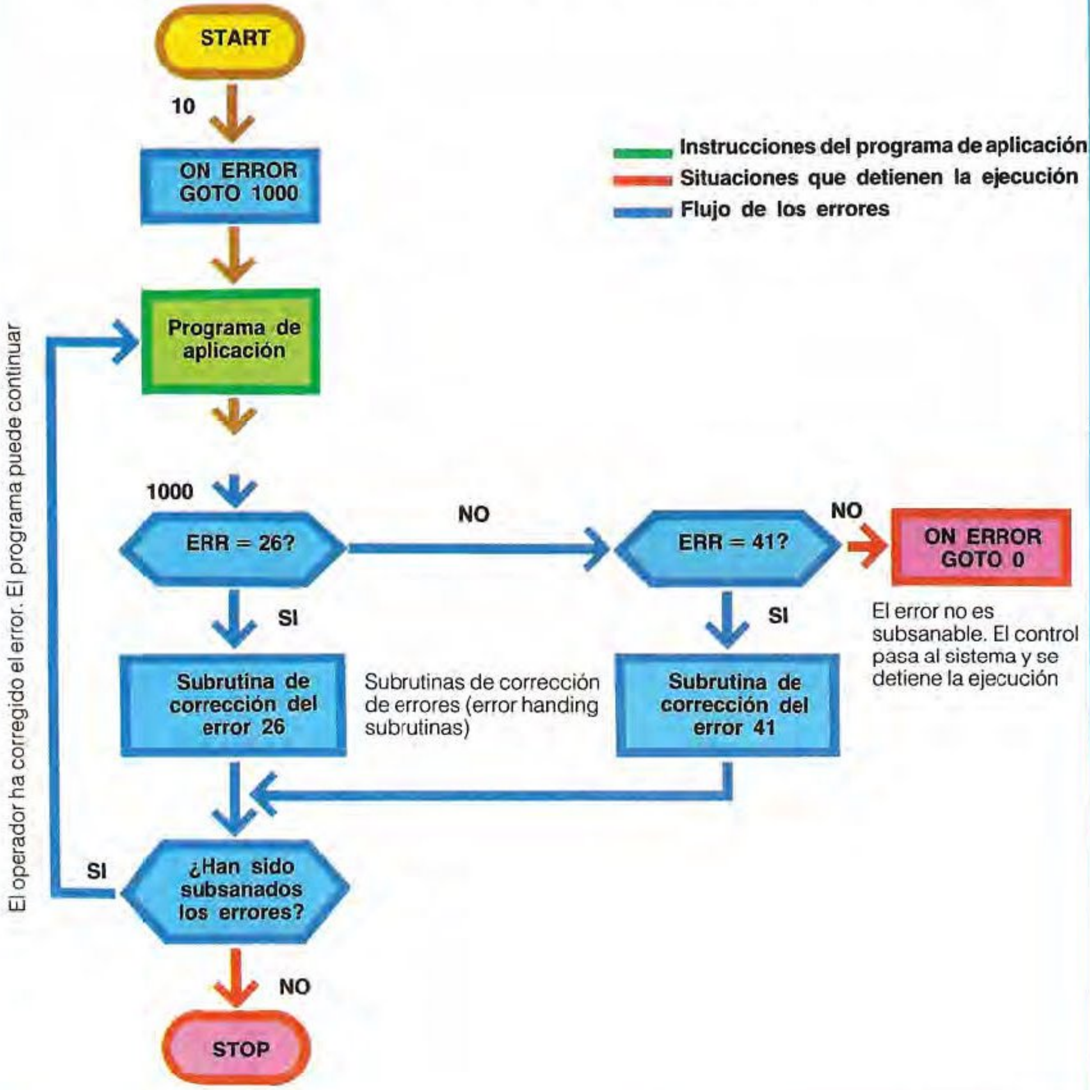
El primer método no presenta dificultad alguna: se trata sólo de efectuar los oportunos controles inmediatamente después de la introducción de datos; si el control da resultado negativo, se vuelve a presentar la petición de introducción.

Por el contrario, el uso de una subrutina generalizada implica una mayor complejidad de los programas, puesto que hay que prever y direccionar todos los errores posibles. A pesar de ello, en los programas de cierta complejidad conviene adoptar este método, si no por otra cosa, para tener todos los diagnósticos concentrados en una única zona del programa. Para ilustrar esta técnica, desarrollemos un programa para la gestión de una agenda de citas.

Los datos están contenidos en un file denominado AGENDA, por orden creciente de fechas.

El formato del registro es el siguiente (ver gráfico inferior de la página contigua):

LOGICA DE CONTROL Y EVENTUAL CORRECCION DE ERRORES



FORMATO DEL RECORD DEL FILE AGENDA



Ha nacido la turismática

«Hotel Automation» es el nombre de una serie de nuevos servicios de telemática para hoteles ofrecidos por la Italtel Telematica en colaboración con la Honeywell Information Systems. La Italtel suministra las centralitas telefónicas y la Honeywell los mini y microordenadores.
Hoy día, la tecnología y la demanda han roto los límites entre la informática y las telecomunicaciones: la colaboración entre las dos empresas mencionadas constituye un importante ejemplo de integración, a través de la telemática, de dos mundos hasta ahora independientes.
Con el nuevo sistema, los hoteles de cualquier tamaño, desde los de gestión familiar hasta los más grandes, pueden ofrecer toda una serie de servicios telefónicos y telemáticos y utilizar la capacidad de elaboración de datos de un ordenador. Los clientes tienen a su disposición, por ejemplo, grabación automática de las llamadas del exterior, identificación y pago mediante tarjetas magnéticas personalizadas y señalización inmediata de eventuales situaciones de emergencia. El hotel dispone de un instrumento de recogida y elaboración de datos que agiliza las operaciones de gestión y contabilidad.
La integración de la centralita y el ordenador no requiere cambios en la red telefónica interna y facilita la conexión con otros servicios, como el télex, el facsímil o la difusión por cable. También se puede conectar, mediante terminales de video, con bancos de datos exteriores al hotel.

El proyecto «Honeywell Turismatica» fue presentado el 21 de febrero de 1983 en la convención «Automazione e Informatica nel Turismo», organizado por la BIT 83 (Borsa Internazionale del Turismo) en colaboración con la Honeywell Information Systems Italia.
El proyecto turismática prevé tanto la propuesta de aplicaciones prácticas basadas en los mini y microordenadores, como la constitución de un comité técnico-científico, con participación de expertos en informática y en turismo, para el estudio de los problemas relativos a la introducción de la informática en las empresas del ramo.*
Todos saben el papel que juega el turismo en la economía italiana y, sobre todo, lo que representa para la balanza de pagos. Según estimaciones oficiales, en 1982 hubo en Italia 103 millones de visitantes extranjeros, con unas entradas de divisas equivalentes a 10,8 billones de liras. Para afrontar las perspectivas de crisis que afectan también al nivel del consumo turístico, es necesario que las estructuras turísticas italianas potencien y racionalicen sus organizaciones mediante instrumentos adecuados, entre los cuales ya es fundamental la informática.
La gestión de un hotel es una cuestión extremadamente compleja y de difícil racionalización; efectivamente, la actividad de un hotel está tipificada por una antigua tradición que determina niveles de servicios y prestaciones que difícilmente se dan ya en otros sectores, pero que en el contexto hotelero representan aún los elementos en base a los cuales se mide la categoría de un establecimiento. Aunque la actividad hotelera está vinculada aún a «modelos de conducta» de origen antiguo, el sector hotelero, como los demás, no puede prescindir de los problemas y objetivos fundamentales de todos los operadores económicos:

- *ajuste de la relación servicio-precio de cara a obtener un máximo de beneficios;*
- *racionalización de la estructura organizativa y de los procedimientos internos para minimizar los costos.*

El logro de estos dos objetivos va ligado, no sólo a la experiencia y la capacidad empresarial de los cuadros directivos, sino también a la utilización de un sistema de recogida y ordenación de datos que permita, a quien ha de tomar las decisiones, aprovechar plenamente la propia experiencia y capacidad.
Concretamente, la inserción de metodologías informáticas en un contexto hotelero ha de permitir, por una parte, una correcta gestión de los datos enviados desde los distintos puntos (habitaciones, bar, restaurante, etc.) y, por otra, que dichos datos constituyan la base sobre la cual el personal del hotel pueda llevar a cabo la gestión de la actividad hotelera.
Especial interés revisten, por otra parte, los problemas de las agencias de viajes. Surgidas como intermediarias entre servicios ajenos, las agencias han ido ampliando cada vez más sus actividades, convirtiéndose a menudo en auténticas «productoras de bienes». En un contexto operativo externo en constante evolución, la agencia de viajes se ve abocada a afrontar problemas muy diversos, entre los cuales:

- *problemas operativos cada vez más complejos en la organización de los viajes propios y en la gestión de las «puntas» de demandas de servicios en temporada;*



Honeywell



Honeywell

Dos imágenes del ordenador personal Honeywell Questar/M9050, instalado en las oficinas de una agencia de viajes.



Proyecto turismática: el ordenador personal Questar/M9050 instalado en la recepción de un hotel.

- *normativas en continua evolución en los sectores fiscal y administrativo;*
- *dificultades para evaluar, en el caso de la organización de viajes propios, la incidencia de los costos y la rentabilidad de las distintas operaciones, tanto en términos globales como por «línea de producto».*

Por lo que respecta a los hoteles, como se desprende de la ponencia sobre «Informática en el turismo: situación actual y perspectivas», presentada en la convención por el profesor Giuseppe Carone, presidente del comité italiano de la Organización Mundial de Turismo (OMT), la introducción de la informática es, en Italia, un fenómeno bastante reciente (no anterior a 1980) y por ahora se limita a los grandes hoteles y en un porcentaje muy bajo (el 4,9% de 5.000 establecimientos de categoría superior, según una encuesta realizada en enero de 1983 por la Honeywell Information Systems Italia).

La situación es algo mejor en el sector de las agencias de viajes, donde la informatización ha comenzado unos años antes (1977).

Según la encuesta citada, la tasa de penetración EDP en enero de 1983 era del 16,6% del total italiano (unas 2.800 agencias).

Tanto en el caso de los hoteles como, en menor medida, en el de las agencias de viajes, hay que señalar, sin embargo, que en buena parte de los casos la aplicación EDP se limita a los procedimientos administrativos normales y no afecta aún a las funciones específicas (check-in, check-out y reservas en el caso de los hoteles; gestión de viajes propios o como intermediarios y gestión de billetes, en el de las agencias de viajes).

Ahora bien, es precisamente el sector de los servicios en general, y de los turísticos en particular, el que, por las características de las actividades desarrolladas, está destinado a obtener de la automatización el máximo de beneficios estructurales y operativos. Sin embargo, es necesario que el cambio tenga lugar en un contexto que no comprometa, sino que revalorice, el capital de competencia y profesionalidad acumulado por los operadores. De ahí la necesidad de comprender en qué consisten las nuevas tecnologías de automatización y cómo pueden aplicarse a la realidad específica del turismo. Éste es precisamente el significado del término «turismática», acuñado para su proyecto por la Honeywell: no se trata de la mera transposición a los hoteles o las agencias de viajes de técnicas y metodologías informáticas ya adoptadas en otros sectores, sino del desarrollo de instrumentos específicos para la racionalización y evolución del servicio turístico. La Honeywell ha presentado a los operadores del sector turístico dos soluciones prácticas basadas en micro y miniordenadores: una para la gestión de hoteles y otra para la de agencias de viajes.

La solución para los hoteles funciona tanto con microordenadores Honeywell Questar/M como con miniordenadores Honeywell DPS 6, ambos disponiendo incluso de varios puntos de trabajo, y se articula sobre cinco módulos:

— reservas
— check-in
— gestión del cliente en el hotel
— check-out
— administración y contabilidad

El módulo de reservas permite gestionar la disponibilidad de habitaciones para dos años, efectuar reservas múltiples (varias habitaciones con una sola reserva), tener en cuenta las preferencias de los clientes, imprimir para los servicios del hotel que la necesiten la lista de llegadas y salidas, etc.

También permite gestionar el llamado «overbooking» y, en temporada baja, concentrar las reservas en determinadas zonas o plantas del hotel para reducir los gastos de calefacción y los servicios de planta.

El módulo de check-in se encarga del registro de los clientes que llegan (si el cliente es habi-

tual, los datos son tomados automáticamente del archivo) y, además de actualizar los diversos archivos, imprime automáticamente los formularios para la policía, evitando que, por olvido o descuido, el trámite quede sin cumplimentar.
El módulo de gestión del cliente en el hotel se ocupa de actualizar en cada momento su cuenta pendiente en función de los servicios que recibe. Se encarga, además, de gestionar los eventuales cambios de habitación y, al marcharse los clientes, actualiza los diversos archivos, entre ellos el de reservas.
El módulo de check-out se encarga de emitir para cada cliente que se marcha hasta cuatro documentos (factura, cuenta, facsímil de factura y facsímil de cuenta), cada uno en distintas lenguas y teniendo en cuenta los acuerdos especiales (el sistema puede gestionar hasta mil acuerdos distintos). Simultáneamente, este módulo actualiza los archivos de contabilidad. Éstos son gestionados por el módulo de administración y contabilidad, que se encarga además de la impresión de los distintos documentos relativos a la gestión del hotel:
— diario de caja
— diario de cuentas pendientes
— situación y resúmenes de las cuentas, etc.
Todo el sistema se caracteriza, por una parte, por una gran facilidad de uso y adecuación a las costumbres hoteleras, teniendo en cuenta que ha de ser utilizado por personal que normalmente no es experto en EDP y que está acostumbrado a un cierto tipo de lenguaje y de comportamiento; por otra parte, posee una gran elasticidad, especialmente necesaria en un ambiente en el que la adaptación a situaciones atípicas es la norma. Así, por ejemplo, en el caso de llegadas o salidas imprevistas, camas añadidas, cuentas separadas para ocupantes de una misma habitación o, viceversa, distintas habitaciones cargadas a una misma cuenta, dicha elasticidad es fundamental.
La solución para agencias de viajes funciona en base a microordenadores Honeywell Questar/M de varios puestos de trabajo, y se articula sobre cuatro módulos:
— gestión de actividades de viaje
— gestión de billetes
— gestión de reservas como intermediario
— contabilidad general
El módulo de gestión de actividades de viaje se ocupa tanto de los medios de transporte como de los hoteles y otros servicios, incluso combinados entre sí para individuos o grupos, con posibilidad de variación de fechas, titulares, servicios, etc., e incluye el cálculo de las cuotas de venta, así como (para los viajes propios al extranjero) de la cuota de divisas por viajero.
El módulo emite los siguientes documentos: vales, confirmación de viajes, listas de viajeros y reservas para los medios de transporte, hoteles y servicios, listas de billetes a emitir. Además se conecta, previa confirmación del operador, con el módulo de contabilidad general para la contabilización de las operaciones y la emisión de facturas proforma y de extractos de cuentas.
El módulo de gestión de billetes se ocupa de los billetes de todo tipo (aéreos, ferroviarios, etc.), tanto en dotación como comprados a terceros. El módulo se encarga de una auténtica gestión de almacén con control de existencias, remanentes mínimos, numeración, etc. Además, gestiona un archivo de billetes con memorización de los datos de emisión y de eventuales rembolsos o anulaciones, y, en el caso de emisión de billetes para actividades propias, con conexión automática con el módulo de gestión de actividades de viaje.
El módulo de gestión de reservas como intermediario se ocupa de las operaciones de reserva en nombre y por cuenta de un cliente relativas a cualquier tipo de servicio nacional o extranjero. Se encarga de la emisión de vales análogos a los emitidos por el módulo de gestión de actividades de viaje (y que, con éstos, confluyen en un único archivo de vales para subsiguientes controles operativos y contables), así como resguardos de consigna, facturas de proveedores dirigidas al cliente y facturas propias al proveedor para las comisiones.
El módulo de contabilidad general, por último, mediante un plano de cuentas personalizable, gestiona la contabilidad clientes/proveedores en partidas abiertas, se encarga de los registros IVA, contabiliza los costos para centros de costo (actividades de viaje), efectúa balances de comprobación y resúmenes, así como los cierres periódicos.
Como ya se ha dicho, la mayor parte de los movimientos contables es activada automáticamente por los demás módulos, lo que permite reducir al máximo el trabajo manual y la posibilidad de error.

---

* Turismática es una marca registrada por la Honeywell Information Systems Italia.

1 / Fecha:
Mes = 2 caracteres (entero comprendido entre 1 y 12)
Día = 2 caracteres (entero comprendido entre 1 y 31)
Hora = 2 caracteres (entero comprendido entre 8 y 20; se consideran sólo las horas diurnas)
Minutos = 2 caracteres (entero comprendido entre 0 y 50 con paso 10)
Para los minutos se consideran sólo las decenas, redondeando a la decena superior

2 / Persona: 30 caracteres (apellido y nombre)
3 / Ciudad y Calle: 40 caracteres
4 / Motivo de la cita: 2 caracteres (sigla)
5 / Comentarios: 50 caracteres

La longitud del file (número de registros que contiene) no se conoce a priori y ha de ser leída en el registro 1. Las funciones que el programa ha de poder llevar a cabo son:

a: impresión de la lista de citas entre dos fechas
b: selección de las citas relativas a una determinada ciudad
c: selección de las citas que tienen un determinado objeto.

Un programa así puede articularse en las siguientes fases principales:

— inicialización
— lectura de los parámetros de selección
— búsqueda e impresión de los datos

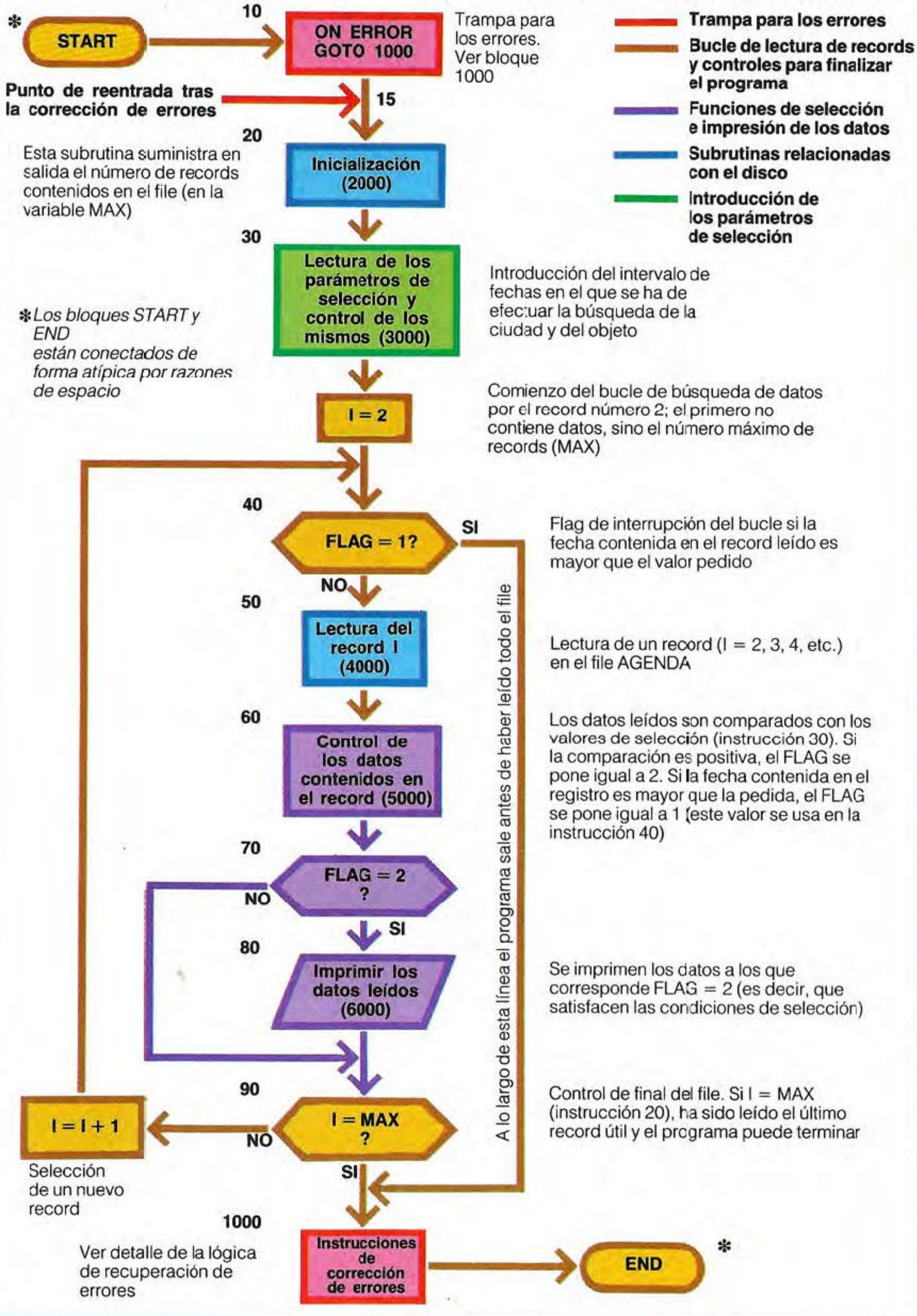
En la fase de inicialización se realizan las funciones de control de existencia del file AGENDA y de lectura de su longitud (contenida en el registro 1; ver, en pág. 231, el capítulo «Los ficheros»). La lectura de los parámetros de selección requiere la introducción del intervalo de fechas, de las ciudades y de los objetos que interesan (estos dos últimos parámetros son optativos).
La última parte prevé la lectura del file y el control, para cada record, de la correspondencia con los datos introducidos; en caso de correspondencia, el record se ha de imprimir, de lo contrario se pasa a elaborar el siguiente record.
En la página contigua se muestra el diagrama de flujo de primer nivel, en el que se prevé una salida antes de que el file sea leído por completo, en caso de que la fecha contenida en el record leído sea mayor que la pedida (FLAG = 1; recordemos que el file AGENDA está ordenado por fechas crecientes).
La lógica de búsqueda es la típica de un bucle

Con la ayuda de un ordenador personal, la gestión de una agenda de citas se vuelve mucho más sencilla.



Honeywell

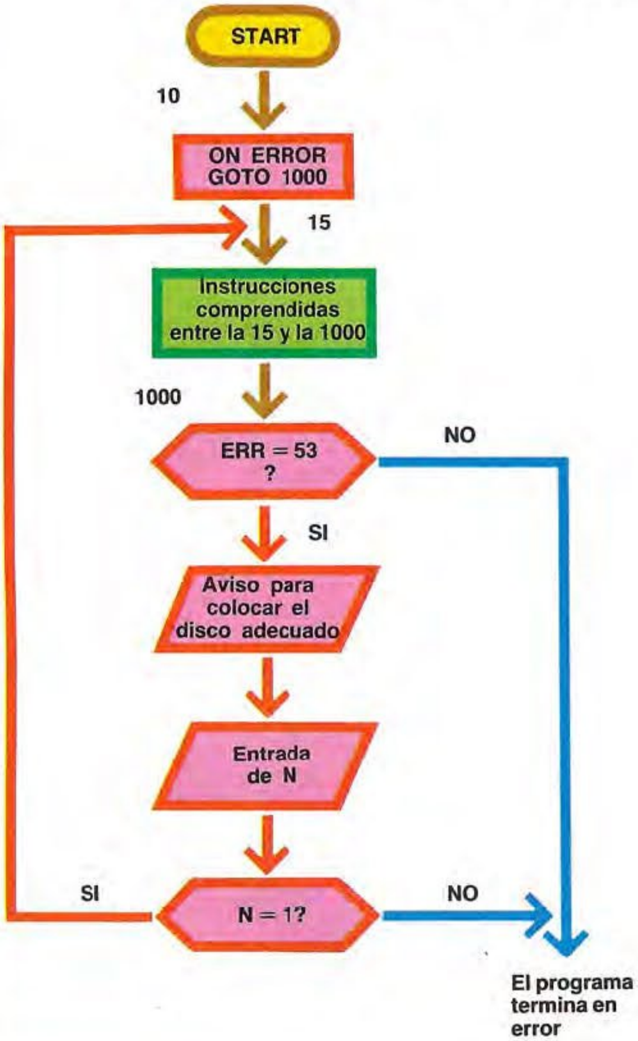
DIAGRAMA DE PRIMER NIVEL DEL PROGRAMA DE GESTION DE AGENDA



con ruptura de código: el file es leído secuencialmente a partir del primer record. Si la fecha contenida en el record leído es inferior a la máxima buscada, la lectura ha de seguir, puesto que puede haber otros records con fechas a seleccionar. Al primer record que contenga una fecha superior a la máxima introducida, se puede interrumpir la búsqueda, ya que todas las demás fechas del file son sin duda superiores.
La ordenación de los records por fechas crecientes puede aprovecharse también para iniciar el escrutinio del file no por el record 2, sino por el que contenga la fecha más próxima a la inferior introducida. A tal objeto se puede utilizar una subrutina de búsqueda dicotómica.

El gráfico inferior muestra el diagrama que incluye las instrucciones de control de errores (expansión del bloque 1000). En este caso, el único error de sistema que puede subsanarse es el de no existencia del file, codificado, a título de ejemplo, como 53. Al verificarse este error se emite un mensaje de advertencia, tras el cual el programa espera a que el usuario finalice las oportunas operaciones correctoras (en este caso, el cambio del diskette). El programa puede ser puesto a la espera insertando una instrucción de input desde la consola; el operador dispone entonces del tiempo para cambiar el disco; después, introduciendo el valor correspondiente, puede reactivar la elaboración.

DETALLE DE LA LOGICA DE CORRECCION DE ERRORES



Ver diagrama de primer nivel del programa de gestión del file AGENDA

El error reconocido corresponde al código 53: no ha sido hallado el file AGENDA

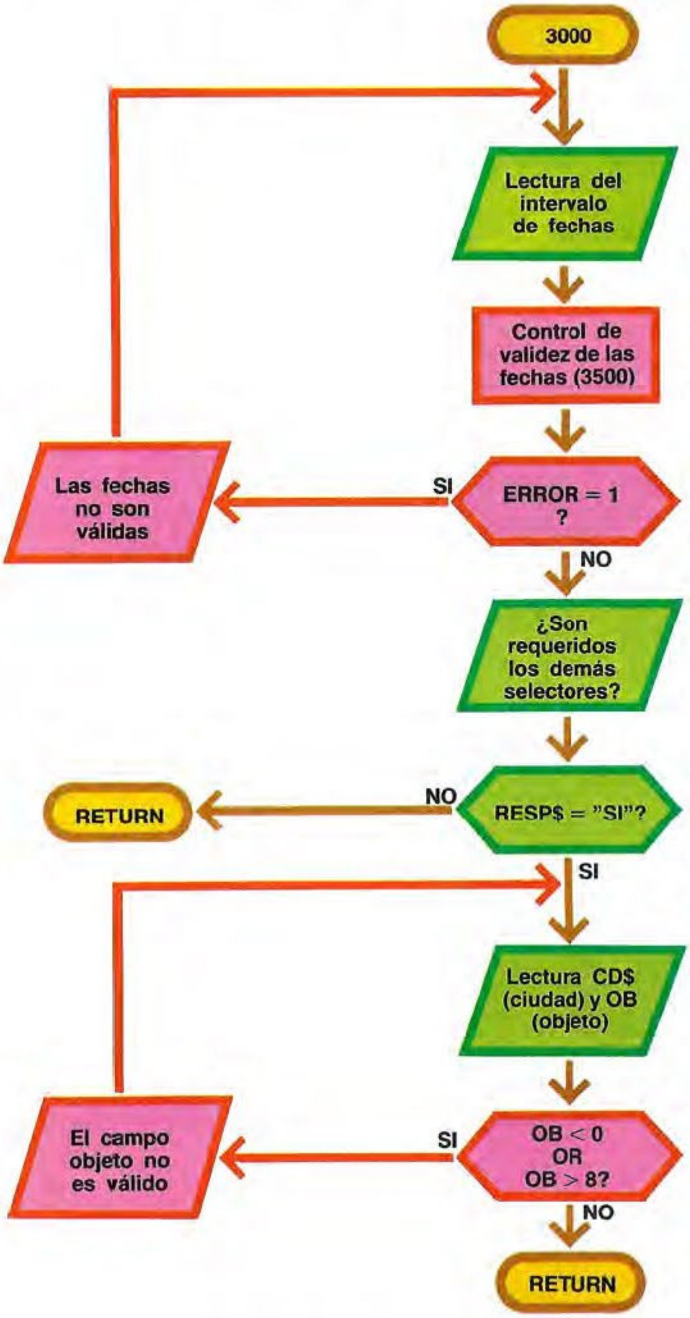
El programa escribe en pantalla un mensaje de advertencia

Para dar tiempo al operador de montar el disco correcto, se requiere la entrada de una variable cualquiera, por ejemplo N. El sistema se pone a la espera de dicha entrada. El operador puede comprobar el disco y, finalmente, decide si prosigue el programa (introduciendo el valor 1) o si lo detiene (introduciendo un valor cualquiera distinto de 1)

DETALLE DE LA SUBRUTINA DE INTRODUCCION DE LOS PARAMETROS DE SELECCION

Comprobación de los datos introducidos (rojo)

Lectura de los datos y decisiones subsiguientes (verde)



Lectura de las fechas entre las cuales hay que efectuar la búsqueda (mes y día de comienzo, mes y día de fin)

Control de validez. Si las fechas contienen un error, la variable ERROR se pone igual a 1

Si hay errores en las fechas se emite un mensaje y se piden nuevos valores

El programa pregunta si se desea la selección también por ciudades y por objetos

La respuesta del operador está en la variable de cadena RESP$. Si su contenido es SI, la subrutina procede a la lectura de los demás parámetros

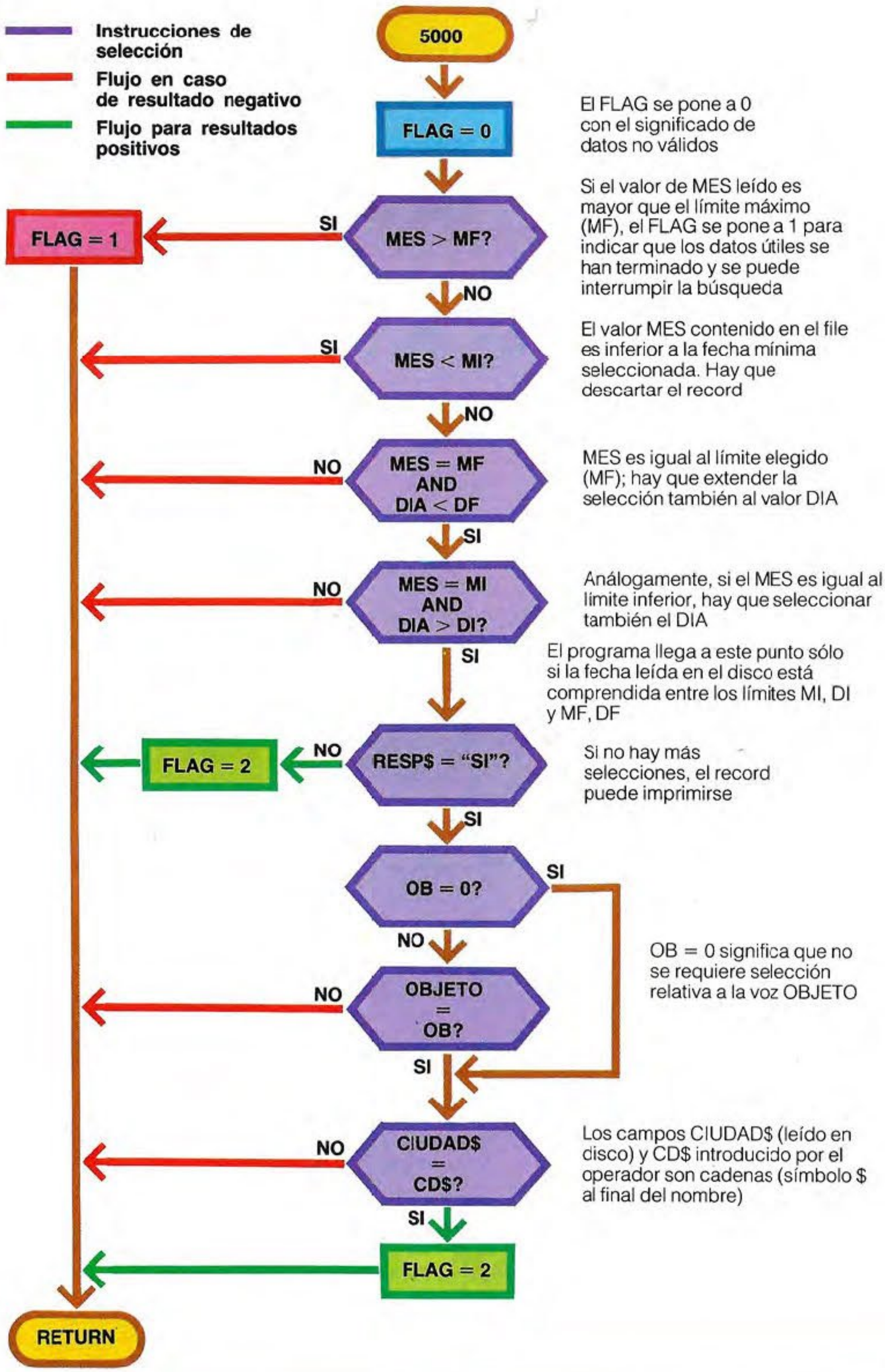
Para RESP$ = "SI", son requeridos tanto la ciudad (CD$) como el objeto (OB)

Suponiendo que los códigos reconocidos sean de 1 a 8, este control determina si hay un error de introducción. El valor 0 es aceptado e indica que no se requiere la selección de la voz OBJETO (sólo se selecciona la ciudad)

La rutina da, en salida, los siguientes parámetros:

DI (día), MI (mes) = fecha en que se inicia la selección
DF (día), MF (mes) = fecha en que finaliza la selección
RESP$ = "SI" si se requieren más selecciones
CD$ = ciudad a seleccionar (nombre de la misma)
OB = objeto (código)

RUTINA DE SELECCION DE DATOS



En la página 427 se muestra el diagrama detallado de la subrutina de introducción de los parámetros de selección (subrutina 3000) y en la pág. 428, la subrutina 5000, dedicada a la selección de los datos leídos en el archivo según los parámetros introducidos.

La subrutina 5000 ha de recibir en entrada los datos MES, DÍA, HORA, MINUTO, CIUDAD, OBJETO, leídos en el file AGENDA, además de los parámetros: fecha de comienzo de la búsqueda, fecha final de la búsqueda, ciudad y objeto, tomados de la subrutina 3000.

PROGRAMA DE AGENDA

```
2 ' ** PROGRAMA DE AGENDA **
4
6 '          MAIN
8 '       FILE = AGENDA
10 ON ERROR GOTO 1000
11 OPTION BASE 1
12 BI$=CHR$(27)+"+"+CHR$(7)
15 '
20 GOSUB 2000          'ESTA SUBRUTINA DA EL N. DE RECORDS CONTENIDOS
25 '                    EN EL FILE (MEMORIZANDOLO EN LA VARIABLE "MAX")
30 GOSUB 3000          'LECTURA DE LOS PARAMETROS DE SELECCION
35 FOR I=2 TO MAX
40 IF FLAG=1 GOTO 1090
50 GOSUB 4000              'LECTURA DEL RECORD (I) DEL FILE "AGENDA"
60 GOSUB 5000              'CONTROL DE LOS DATOS CONTENIDOS EN EL RECORD
70 IF FLAG<>2 GOTO 90
80 GOSUB 6000              'IMPRIMIR LOS DATOS LEIDOS
90 NEXT I
1000 ' *INSTRUCCIONES DE CORRECCION DE ERRORES
1010 IF ERR<>53 THEN LPRINT    "TIPO DE ERROR NO PREVISTO":STOP
1020 PRINT     "ERROR EN DISCO: MONTAR EL CORRESPONDIENTE AL PROGRAMA"
1030 LPRINT     "ERROR EN DISCO: MONTAR EL CORRESPONDIENTE AL PROGRAMA"
1040 PRINT     "INTRODUCIR (1) PARA PROSEGUIR"
1050 PRINT     "INTRODUCIR (2) PARA DETENER EL PROGRAMA"
1060 INPUT;N
1070 IF N=1 GOTO 15
1080 PRINT     "EL PROGRAMA TERMINA EN ERROR": STOP
1090  END
1100 '
1110 '
1120 '
1130 '
2000 ' ** SUBRUTINA DE PRUEBA: SE INSERTARA Y EXPLICARA MAS ADELANTE **
2010 '
2020 MAX=5     'EJEMPLO DE PRUEBA
2030 RETURN
2040 '
2050 '
2060 '
2070 '
2080 '
3000 ' ** SUBRUTINA DE LECTURA DE LOS PARAMETROS DE SELECCION **
3010 PRINT BI$
3020 PRINT      "INSERTAR EL INTERVALO DE FECHAS ENTRE LAS QUE EFECTUAN LA BUSQUEDA"
3030 PRINT      "EN LA FORMA: MES Y DIA INICIAL, MES Y DIA FINAL (MM/DD)"
3040 INPUT      "MES"; MI
3050 INPUT      "DIA"; DI
3060 INPUT      "MES"; MF
3070 INPUT      "DIA"; DF
3500 ' ** RUTINA DE CONTROL DE VALIDEZ DE LAS FECHAS **
3510 ERROR=0      'INICIALIZA LA VARIABLE "ERROR"
3520 IF MI>12 OR MF>12 THEN PRINT "LAS FECHAS NO SON VALIDAS": ERROR=1
3530 IF DI>31 OR DF>31 THEN PRINT "LAS FECHAS NO SON VALIDAS": ERROR=1
3535 IF MF<MI THEN PRINT            "LAS FECHAS NO SON VALIDAS": ERROR=1
3540 IF ERROR=1 GOTO 3020
3600 PRINT BI$
3610 PRINT    "¿SE DESEA LA SELECCION TAMBIEN POR CIUDAD Y OBJETO? (SI/NO)"
3620 INPUT  RESP$    'LA CADENA "RESP$" CONTIENE LA RESPUESTA (SI/NO)
3630    IF RESP$<>"SI"THEN RETURN
3640 INPUT "CIUDAD          "; CD$
3645 PRINT "EL OBJETO ESTA EN UN CODIGO NUMERICO COMPRENDIDO ENTRE 1 y 8"
3650 INPUT "OBJETO          "; OB
3660 IF OB<0 OR OB>8 THEN PRINT    "EL CODIGO (OBJETO) NO ES VALIDO": GOTO 3650
3670 RETURN
```

```
4000 ' ** SUBRUTINA PARA LA LECTURA DEL RECORD (I) DEL FILE AGENDA **
4010 '
4020 ' Esta rutina se insertará y explicará más adelante
4030 PRINT BI$
4040 PRINT  "ESTA RUTINA ESTA SIMULANDO UNA LECTURA EN DISCO"
4050 PRINT  "INSERTAR LOS DATOS REQUERIDOS PARA HACER UNA PRUEBA"
4060 INPUT     "MES    ";MES
4070 INPUT     "DIA    ";DIA
4080 INPUT     "OBJETO";OBJETO
4090 INPUT     "CIUDAD";CIUDAD$
4100 RETURN
4110 '
4120 '
4130 '
4140 '
4150 '
5000 ' ** SUBRUTINA PARA EL CONTROL DE LOS DATOS CONTENIDOS EN EL RECORD (I) **
5010 '
5020 FLAG=0
5030 IF MES>MF THEN FLAG=1:PRINT "LA FECHA NO ESTA EN CAMPO DE BUSQUEDA": GOTO 5130
5040 IF MES<MI THEN:PRINT       "LA FECHA NO ESTA EN CAMPO DE BUSQUEDA": GOTO 5130
5050 IF MES=MF AND DIA>DF THEN :PRINT "LA FECHA NO ESTA EN CAMPO DE BUSQUEDA": GOTO 5130
5060 IF MES=MI AND DIA<DI THEN :PRINT "LA FECHA NO ESTA EN CAMPO DE BUSQUEDA": GOTO 5130
5070 IF RESP$ <>"SI" THEN FLAG = 2: GOTO 5130
5080 IF OB= 0 GOTO 5100
5090 IF OBJETO <>OB THEN PRINT "EL OBJETO NO COINCIDE CON EL QUE
                                 "HAY QUE SELECCIONAR": GOTO 5130
5100 IF CIUDAD$<> CD$ THEN PRINT "LA CIUDAD NO COINCIDE CON LA QUE HAY QUE
                                    " SELECCIONAR": GOTO 5130
5110 FLAG=2
5120 RETURN
5130 PRINT "INTRODUCIR UN CARACTER PARA CONTINUAR"
5140 ' S$ = INPUT $ (1)
5150 ' RETURN
5160 '
5170 '
5180 '
5190 '
5200'
6000' * INSTRUCCIONES DE IMPRESION *
6010 LPRINT        "INTERVALO DE FECHAS ENTRE LAS QUE SE EFECTUA LA BUSQUEDA"
6020 LPRINT        "DEL:";DI;"/";MI;"AL:";DF;"/";MF
6030 LPRINT        "DATOS LEIDOS EN DISCO"
6040 LPRINT        "MES:"; MES
6050 LPRINT        "DIA:";DIA
6060 LPRINT        "CODIGO DEL OBJETO:";OBJETO
6070 LPRINT        "CIUDAD:";CIUDAD$
6080 LPRINT
6090 RETURN
```

```
INTERVALO DE FECHAS ENTRE LAS QUE SE EFECTUA LA BUSQUEDA
DEL:2/2 AL: 30/4
DATOS LEIDOS EN DISCO
MES:3
DIA:15
CODIGO DEL OBJETO:1

CIUDAD: MADRID

INTERVALO DE FECHAS ENTRE LAS QUE SE EFECTUA LA BUSQUEDA
DEL:1/1 AL: 30/3
DATOS LEIDOS EN DISCO
MES:1
DIA:31
CODIGO DEL OBJETO:2
CIUDAD: VALENCIA
```

En la subrutina 5000, la comparación entre las dos cadenas CIUDAD$ y CD$ se efectúa de forma inmediata (IF CIUDAD$ = CD$ GOTO...). Este método simplificado puede encubrir ciertas posibilidades de error. La forma más general y segura será ilustrada más adelante.

La salida de la subrutina 5000 es el parámetro FLAG puesto en los valores:

0 = selección no lograda
1 = fecha del file superior a la máxima seleccionada; interrumpir la búsqueda
2 = selección lograda; datos a imprimir

En la tabla de la pág. 429 (programa principal: diagrama de la pág. 425; subrutina 3000: diagrama de la pág. 427; subrutina 5000: diagrama de la pág. 428) se muestra el listado de las instrucciones y un ejemplo de ejecución. Las partes del programa que faltan (subrutina del disco, introducciones e impresiones) se verán más adelante. Este programa, junto con otro que se mostrará (introducción de datos), es un ejemplo completo de gestión de una agenda.

El hombre frente a la máquina

Tras haber adaptado durante mucho tiempo al hombre a la máquina, hoy se empieza a prestar más atención al «software» humano. ¿Con qué problemas y resultados?

La ergonomía se define como «el arte y la ciencia de proyectar sistemas capaces de prevenir o reducir sensaciones dolorosas, físicas o mentales, reales o imaginarias...» Estudia el confort, la salud, la seguridad y el equilibrio mental de quien vive en un ambiente de trabajo, y ve la relación entre el hombre y la máquina como una guerra a ganar con consideraciones fisiológicas, psicológicas y técnicas.

Si los primeros intentos de automatizar las tareas de oficina encontraron cierta resistencia, se debe en parte al hecho de que las relaciones del hombre con el ordenador fueron inicialmente de antagonismo más que de cooperación. Efectivamente, se pensaba que tenían que ser los operadores quienes se adaptaran a aquellos ambientes climatizados, carentes de polvo y de humor, tan adecuados para las máquinas. Y desde aquel momento comenzó a insinuarse una cierta desconfianza.

Aunque en las tareas de oficina los efectos de la investigación ergonómica comienzan a apreciarse sólo ahora, el escollo más peligroso ya ha sido superado, puesto que la tecnología avanzada ya no se contempla como un enemigo a vencer, sino como un aliado que puede ayudarnos a trabajar como seres humanos.

En la elaboración electrónica, casi todos los problemas físicos han encontrado ya una solución directa e inmediata. Las esquinas redondeadas de las máquinas de oficina son agradables a la vista y menos peligrosas. El funcionamiento de la mano o del ojo es una cuestión de biofísica a la que el hardware puede adaptarse fácilmente. ¿Qué pasa si un dedo golpea miles de veces al día una tecla de plástico? ¿Cómo reacciona una retina, acostumbrada a caracteres negros sobre páginas blancas, si ha de contemplar durante horas una pantalla luminosa?

Las radiaciones ya no son un problema: el NIOSH (National Institute of Occupational Safety and Health) y la Academia Americana de Ciencias han llegado a la conclusión de que el nivel de radiaciones emitidas por las pantallas es demasiado bajo como para provocar cataratas, daños genéticos o perjuicios para la salud. Otros problemas no han sido fáciles de resolver. Una investigación, llevada a cabo también por el NIOSH, ha recogido datos cuantificables sobre distintas molestias, reales o imaginarias.

La causa no está siempre en la pantalla: muchas de estas molestias pueden ser provocadas por una mala iluminación, un asiento inadecuado o un ambiente psicológicamente hostil. Efectivamente, la solución de un problema relacionado con el uso de máquinas a menudo hace aparecer otros, y en consecuencia los proyectistas han de dirigir su atención a toda la esfera ambiental, que no sólo afecta al cuerpo sino también a la mente.

James Martin, autor del clásico Desing of Man-Computer Dialogues, *sostiene que cada tipo de usuario requiere un interface hombre-máquina diferente. Efectivamente, los hombres pertenecen a tipos heterogéneos, no siempre capaces de «interfacearse» entre ellos, y ante un ordenador pueden incluso pasar de un tipo a otro.*

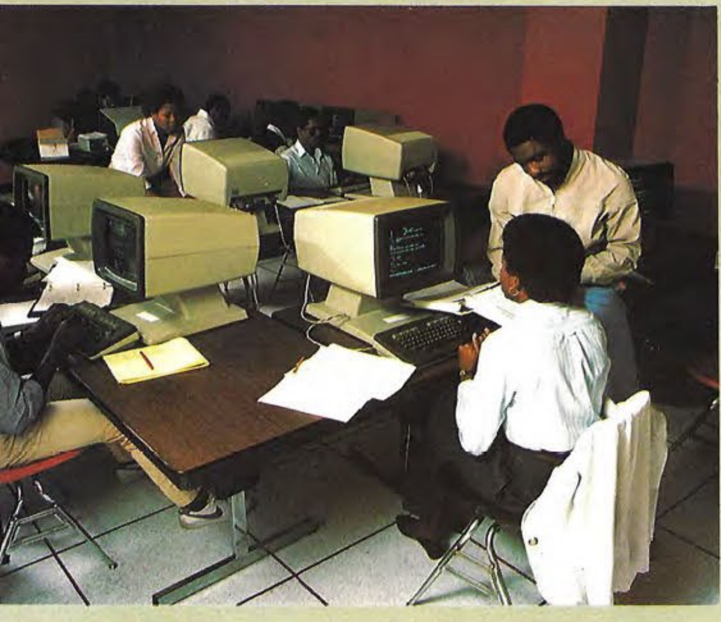
Martin los divide en parejas: asiduo/esporádico, experto/inexperto, inteligente/no inteligente, activo/pasivo, intermediario/interceptor, y sostiene que para cada tipo de usuario hay que encontrar el tipo ideal de comunicación: alfanumérica, sonora o gráfica.

Martin descarta de entrada el lenguaje hablado porque requeriría un software demasiado complicado, y tampoco muestra demasiada simpatía por los lenguajes de programación específicos, como el COBOL y el FORTRAN, porque requieren niveles de profesionalidad y de motivación demasiado elevados para la mayoría de los usuarios. En un artículo publicado en 1982, Paul Heckel ha propuesto una forma distinta de clasificar a los usuarios de ordenadores, sosteniendo que un buen proyectista de software ha de conocer a fondo a su público. «Todos hablamos nuestra lengua madre, pero cada cual tiene su dialecto y, según la profesión, una jerga rica en expresiones típicas de una determinada actividad. Para desarrollar una buena aplicación no basta con conocer la lengua en general, sino también la jerga de quien ha de usarla.» En otras palabras, no se puede olvidar que personas dedicadas a profesiones distintas tienden a pensar de forma distinta.

«Muchas veces nos han pedido que hiciéramos programas que pudieran ser utilizados por todos, incluso por personas carentes de toda preparación. ¡Ojalá hubiera "tablas rasas" así! Por el contrario, la mayoría de nuestros problemas

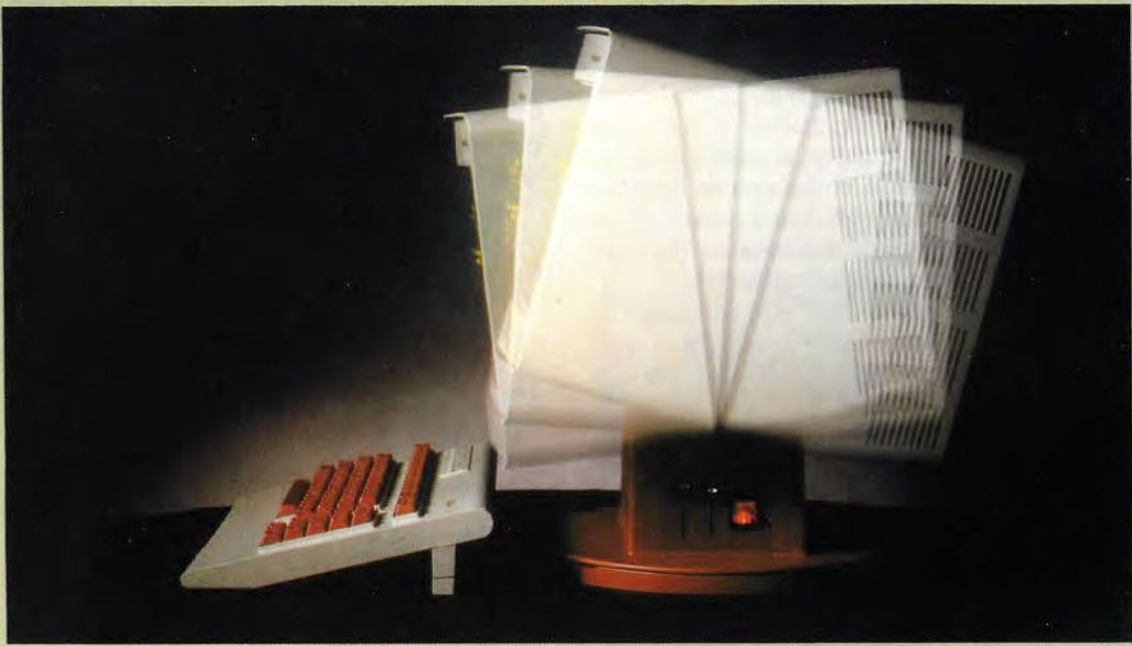
surge de haber pensado implícitamente que el usuario no tiene nada mejor que hacer que despojarse de todos los conocimientos trabajosamente adquiridos en el ejercicio de su profesión para adaptarse a otra jerga.» Hasta la jerga, espanto de puristas, puede facilitar el acercamiento al nuevo usuario. Si por una lengua madre se han hecho revoluciones, bien se podrán conquistar empleados, secretarias, abogados o médicos utilizando en pantalla un lenguaje que remita a sus orígenes y tradiciones. Por el contrario, un buen programa puede ser malogrado por una jerga que describa los procesos internos de la máquina pero que, en cambio, ignore los de la caja craneana. En otras palabras: es preferible un programa tal vez algo renqueante en su lógica pero escrito en el lenguaje del usuario, a un programa técnicamente perfecto dedicado a la solución de un problema erróneo.
Un manual, una página de vídeo o un menú mal escritos pueden hacer más daño a la mente humana que el que pueda causarle a la mano un ángulo mal diseñado del teclado. Un cursor que parpadea puede esperar teóricamente por toda la eternidad la pulsación de la oportuna tecla. Pero para la mente humana el asunto es distinto. Según la Convergent Technologies, empresa de proyectos ergonómicos, un operador no suele estar dispuesto a esperar una respuesta durante más de segundo y medio. Por otra parte, en Man-Computer Dialogues, Martin sostiene que si el tiempo de respuesta es demasiado breve, el usuario inexperto sospecha que los datos no son válidos.
*La efímera duración de la atención humana constituye uno de los aspectos más críticos de un interface hombre/máquina. Últimamente se ha discutido ampliamente sobre la conveniencia de usar menús en lugar de sentencias. Con los menús, el usuario sólo tiene que elegir entre un determinado número de alternativas, sin necesidad de recordar todas las sentencias, puesto que el ordenador se encarga automáticamente de proponerle una lista con las posibles elecciones. De esta manera, al no haber esfuerzo memorístico, los tiempos de aprendizaje son mucho más breves. Pero si van bien para los principiantes, los menús resultan terriblemente tediosos para los usuarios más expertos, obligados a tragarse una página tras otra de elecciones que ya se saben de memoria. Por este motivo, un buen paquete * ha de incluir varias rutinas de bypass, de forma que los usuarios más duchos puedan tomar atajos. Hasta los videojuegos del tipo Dragons & Dungeons tienen en cuenta este principio, mientras que un costosísimo paquete de contabilidad empresarial puede provocar colapsos impresionantes.*
Cuando hablamos con un desconocido intentamos, instintivamente, evaluar su nivel intelectual: ¿comprende lo que estamos diciendo? ¿Toma parte activa en la discusión? Nos hacemos, así, una opinión de él que nos induce a adoptar un determinado tono y nivel de lenguaje. En la práctica, terminaremos comportándonos como un maestro, como un colega o como un alumno. ¿Por qué un paquete de software no podría reaccionar de la misma manera? Según Gerry Gassman, responsable de la ergonomía de los sistemas de gestión de Hewlett Packard, hay que preocuparse ante todo de definir con mucho cuidado el contexto en que el nuevo instrumental será utilizado, puesto que cada ambiente requiere un tratamiento distinto. Por ejemplo, en un centro de elaboración electrónica o en un área de servicio, los operadores trabajan estando casi siempre de pie. Los paneles de mando

Disposición ergonómica de los terminales en un centro de cálculo de la empresa Hewlett Packard.



K. Reese/Marka

* Los paquetes son grupos de programas de aplicación, orientados hacia la solución de problemas de gestión o científicos concretos, desarrollados por personal especializado y puestos a la venta como accesorios de los sistemas de elaboración.



Un proyecto correcto ha de prever la posibilidad de inclinar el monitor.

han de estar, por lo tanto, dispuestos hacia adelante y hacia arriba para que resulten fácilmente visibles y alcanzables. Si, por el contrario, el usuario está sentado ante un escritorio, los indicadores pueden estar más lejos, siempre que queden a la altura de los ojos, pero los paneles de control y las unidades de I/O han de quedar al alcance de la mano.

En su metodología de proyecto, Hewlett Packard sigue luego con un análisis de las modalidades de utilización, encomendado a expertos de diseño y psicología industrial, con una serie de entrevistas a usuarios potenciales, con la construcción de modelos y con experimentos realizados con usuarios no prevenidos. Según Gassman, en este campo Hewlett Packard ha adaptado expresamente una política basada en el respeto de normas más estrictas que las oficialmente impuestas, para poder estar siempre en vanguardia y no correr el riesgo de que una nueva normativa le pille a contrapié.

Aunque en el plano técnico ya han sido resueltos numerosos problemas ergonómicos, en la práctica queda mucho por hacer. En Estados Unidos se confía principalmente en las leyes del mercado, en la convicción de que las máquinas bonitas y cómodas se venden mejor. En Europa se tiende, por el contrario, a legislar. Por ejemplo, en octubre de 1980, el Instituto de seguros contra accidentes de trabajo de Alemania Federal ha promulgado normas de seguridad que estipulan numerosos requisitos a cumplir por los aparatos de oficina. Desde el 1 de enero de 1982, gran parte de estas pautas ergonómicas son normas legales, con lo que la RFA se ha convertido en el primer país del mundo con una legislación en materia de ergonomía.

La CBEMA (Computer and Business Equipment Manufacturers Association), que se ocupa de este problema con una óptica de alcance mundial, concuerda, en general, con la conveniencia de una normativa, pero manifiesta ciertas dudas sobre la validez técnica y la flexibilidad de las pautas de referencia. En todos los países, y no sólo en Estados Unidos, la ergonomía tiende a quedar anclada a posiciones de tipo defensivo, desde las que se pueden detectar fallos y peligros y aliviar daños físicos, pero no potenciar la creatividad.

Hay, sin embargo, una rama de la investigación ergonómica que se ocupa del aprendizaje interactivo y que, mirando un poco por encima de las artrosis cervicales, alcanza a entrever las cabezas. Así, emergen nuevos conceptos ergonómicos, que no sólo se proponen poner remedio a nuestras flaquezas, sino también avivar la chispa del genio.

(Extraído de DATA MANAGER, septiembre de 1983; © INTERACT, enero-febrero de 1983.)

Las funciones

Se llama **función** a cualquier ley que relaciona entre sí dos o más magnitudes. El caso más simple y de uso más difundido es el de las funciones de una variable, en las que las magnitudes relacionadas son sólo dos: a cada valor de una de las dos, la ley de dependencia (función) hace corresponder un valor de la otra.
Por ejemplo, se puede decir que el número de kilómetros que un automóvil puede llegar a recorrer con un litro de combustible es una función de la velocidad.
Supongamos que queremos obtener la ley de dependencia de los kilómetros recorridos con respecto a la velocidad. En ese caso tendremos que echar en el depósito una cantidad de combustible conocida (por ejemplo, 1 litro) y medir cuántos kilómetros recorremos manteniendo una determinada velocidad.

Repitiendo la prueba a distintas velocidades y midiendo los kilómetros recorridos, siempre con 1 litro de gasolina, tendremos el recorrido en función de la velocidad. Durante la prueba podemos elaborar una tabla con las velocidades y los kilómetros recorridos:

Velocidad km/h	**Recorrido (km con 1 litro)**
40	16
60	14
80	12
100	11
120	10

Esta tabla contribuye a cuantificar la ley buscada, pero no basta para evidenciar todas las características de la función que liga la distancia recorrida con la velocidad.

Manual del perfecto terminal

Los terminales han sido quizá los productos más estudiados ergonómicamente. He aquí algunos criterios y requisitos recogidos de diversas fuentes:

1. Los operadores, que al fin y al cabo sólo son seres humanos, adoptan distintas posiciones cuando trabajan con un terminal; después de varias horas de proceso de textos o trabajos similares, los hombros tienden a curvarse ligeramente hacia adelante y las cabezas se inclinan unos 20° con respecto a la vertical. Teniendo en cuenta que, de vez en cuando, la gente cambia de posición y «se estira», el ángulo visual medio de una pantalla debería hallarse entre los 10° y los 40° respecto al plano horizontal. Además, las pantallas han de ser inclinables, para que se puedan adaptar también a los cambios de las condiciones de iluminación.

2. La distancia entre el ojo y la pantalla ha de ser de unos 52 cm y, puesto que un continuo cambio de la distancia focal causaría una seria fatiga ocular, también el pupitre y el teclado han de hallarse más o menos a la misma distancia. En un sistema realmente bien resuelto, el operador tendría que regular sólo una de estas distancias, mientras que las otras deberían adaptarse automáticamente.

3. Todos los controles necesarios han de hallarse en un arco descrito por el brazo semiflexionado.

4. Los ventiladores para la refrigeración de las unidades de disco y de los terminales pueden provocar sequedad en la córnea. Los flujos de aire caliente no han de ir dirigidos hacia membranas mucosas, ni los de aire frío hacia articulaciones.

5. Para evitar que el operador adopte posturas erróneas conviene que el pupitre sea regulable según las exigencias individuales, de forma que sea fácil escribir sobre él, que pueda inclinarse entre 15° y 75° respecto a la vertical, que tenga superficie antirreflejos y que esté inmediatamente detrás del teclado.

6. Las pantallas con caracteres verdes o amarillos cansan mucho menos la vista. Los primeros son preferibles frente a luces fuertes; los segundos, con luces difusas. La elección depende, pues, de las condiciones del local y de los gustos personales.

7. Algunos sostienen que el pasar del negro-sobre-blanco del papel al blanco-sobre-negro de la pantalla fatiga las pupilas, obligándolas a una continua adaptación. Otros dicen que las pantallas (que emiten luz) tienen un efecto distinto que el papel (que la refleja): la retina tendería a «llenar» las zonas oscuras alrededor de los caracteres luminosos, con un efecto de «nivelación» que anularía el contraste.

8. El parpadeo que aparece en la pantalla cuando los caracteres empiezan a desenfocarse, resulta especialmente molesto y puede cansar la mente, además de la vista. Se hace, además, más evidente con el aumento de la luminosidad total. Por este motivo casi todos los terminales utilizan caracteres claros sobre fondo oscuro, con una frecuencia de «refrescado» de 50-60 veces por segundo.

9. Un reflejo fuerte puede volver ilegible la pantalla y causar dolor de cabeza. Para reducirlo se pueden utilizar cristales grabados, filtros, revestimientos o pantallas regulables. Los cristales grabados y los filtros a menudo hacen borrosos los caracteres; los revestimientos son costosos; por el contrario, una pantalla regulable puede resolver el problema.

Definición de una función por puntos

Para analizar más fácil y detalladamente el comportamiento de la ley de dependencia conviene transformar la tabla antes obtenida en un gráfico (**diagrama por puntos de la función**).
A tal objeto, habrá que establecer una escala de velocidades (por ejemplo, haciendo que 1 cm represente 20 km/h) y una escala de distancias recorridas (5 mm representan 2 km recorridos con 1 litro de combustible). En base a estas convenciones, la velocidad de 40 km/h vendrá representada por un segmento de 2 cm, y el recorrido correspondiente (16 km) por un segmento de 40 mm.
Para construir el gráfico tendremos que traducir la tabla en centímetros para las velocidades y en milímetros para los recorridos.
Llamando y al valor de los recorridos en milímetros y x al valor de la velocidad en centímetros, tendremos:

Velocidad km/h	**Traduc. en cm**	**Recorr. km**	**Traduc. en mm**
40	2	16	40
60	3	14	35
80	4	12	30
100	5	11	27.5
120	6	10	25

Llevando los valores de los centímetros sobre una horizontal y los de los milímetros sobre una vertical, se puede dibujar el diagrama de la pág. 436. La recta horizontal (x) en la que van los valores equivalentes a las velocidades se denomina **eje x** o **de abscisas**; la recta vertical (y) en la que se indican las distancias recorridas es el **eje y** o **de ordenadas**.
La representación gráfica de la relación recorrido/velocidad se ha obtenido, como hemos dicho, poniendo 1 cm = 20 km/h y 5 mm = 2 km

10. El contraste es otro factor a tener en cuenta: las normas alemanas sugieren una relación comprendida entre 3:1 y 15:1. Un buen terminal ha de permitir al operador regular el contraste en función de las condiciones de luz y de sus preferencias personales.

11. La postura menos fatigosa para quien ha de teclear durante horas es, probablemente, con los codos a 90° y las muñecas a menos de 10° respecto al plano horizontal, lo que significa teclados planos (y, naturalmente, separables).

12. Según los alemanes, el teclado ideal no ha de ser mucho más alto de 3 cm. Hasta el momento no son muchos los teclados tan planos: pero para adaptar mejor la máquina al hombre, siempre cabe poner un apoyamanos de unos 5 cm delante del teclado.

13. También el ángulo del teclado es un factor clave para reducir la fatiga de los brazos, las muñecas y los codos. Sobre esto, las opiniones son algo más confusas: los alemanes recomiendan un ángulo inferior a 15°, mientras que otros centros de investigación afirman que un ángulo inferior a 7° provoca una elevada tasa de errores y de malestar físico.

14. Para algunos requisitos del teclado hay que distinguir entre operadores expertos e inexpertos. Para quien ha de teclear mirando el teclado, el tener la pantalla a la altura de los ojos puede ser un inconveniente. Los operadores expertos trabajan mejor con un teclado no demasiado «reactivo», mientras que los principiantes prefieren oír el «clic» de cada tecla. En cualquier caso todos concuerdan en que un mínimo de feedback es necesario, para evitar saltarse una pulsación o repetirla dos veces.

15. El «rollover» es la capacidad de aceptar aumentos de velocidad en la digitación de palabras o párrafos que al operador le resultan familiares; por momentos puede llegar hasta 200 ó 300 palabras por minuto. No debe confundirse con la capacidad de memorizar los datos introducidos mediante el teclado mientras es elaborada la línea anterior.

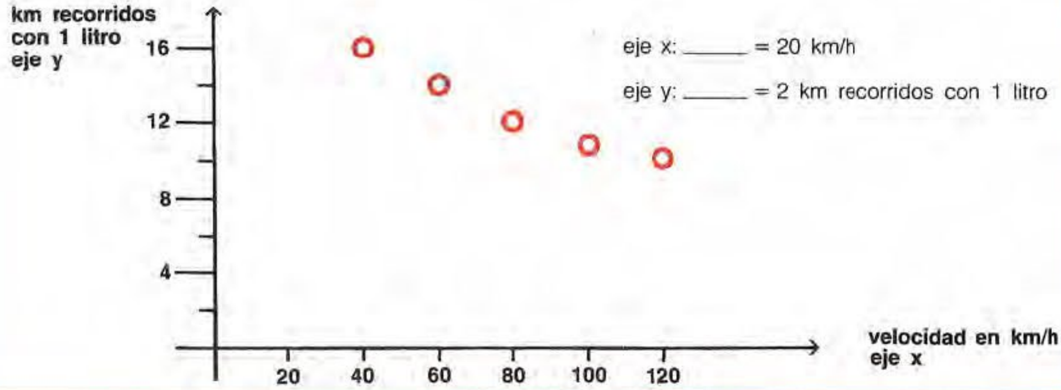
16. La disposición estándar del teclado ya es, de hecho, la de la IBM Selectric. Cualquier variación no hace más que confundir y ralentizar al operador durante semanas, hasta que se acostumbre a la nueva disposición. La única excepción la representa la tecla fijamayúsculas, concebida en su momento para los telescribientes, pero que actualmente, en ambiente DP, ya no tiene ningún valor práctico; por tanto, cabe sustituirla por una tecla de bloque del shift.

17. El teclado se articula en tres áreas. El área primaria contiene los caracteres alfanuméricos y unos pocos comandos de uso más frecuente. Las secundarias, que el operador experto puede alcanzar sin apartar los ojos de la pantalla, normalmente se reservan a las teclas de control de la misma. Las terciarias, que requieren movimientos de los ojos, de las manos y de los brazos, se utilizan para otras funciones eventuales. Esta subdivisión depende también del tipo de operador: quien se ocupe del proceso de textos tendrá preferencias muy distintas que el contable que sólo usa el teclado numérico. En cualquier caso, la extensión de los dedos ha de limitarse al mínimo indispensable, teniendo en cuenta que los principiantes son a menudo «perezosos» y tienden a alcanzar todas las funciones con una sola mano.

18. Los alemanes aconsejan un coeficiente de reflexión del teclado comprendido entre el 20 y el 50 por ciento para reducir la fatiga ocular.

(Tomado de DATA MANAGER, n.° 26, septiembre de 1983.)

TRAZADO POR PUNTOS DE LA FUNCION RECORRIDO-VELOCIDAD



de recorrido. Estos valores (1 cm y 5 mm) son factores característicos, respectivamente, del eje x y del eje y, que permiten transformar las variables en longitudes a señalar sobre los respectivos ejes.

El cálculo de la longitud a considerar, dado el valor de la variable, es inmediato, en base a las siguientes proporciones:

$$\left(\begin{array}{c}\textbf{longitud}\\ \textbf{eje x}\end{array}\right): \textbf{valor x} = \textbf{5:2} \text{ (5 mm = 2 km)}$$

$$\left(\begin{array}{c}\textbf{longitud}\\ \textbf{eje y}\end{array}\right): \textbf{valor y} = \textbf{1:20} \text{ (1 cm = 20 km/h)}$$

$$\left(\begin{array}{c}\textbf{longitud}\\ \textbf{eje x}\end{array}\right) = \frac{\textbf{5} \times \textbf{valor x}}{\textbf{2}} = \frac{\textbf{5}}{\textbf{2}} \times \textbf{valor x}$$

$$\left(\begin{array}{c}\textbf{longitud}\\ \textbf{eje y}\end{array}\right) = \frac{\textbf{1} \times \textbf{valor y}}{\textbf{20}} = \frac{\textbf{1}}{\textbf{20}} \quad \textbf{valor y}$$

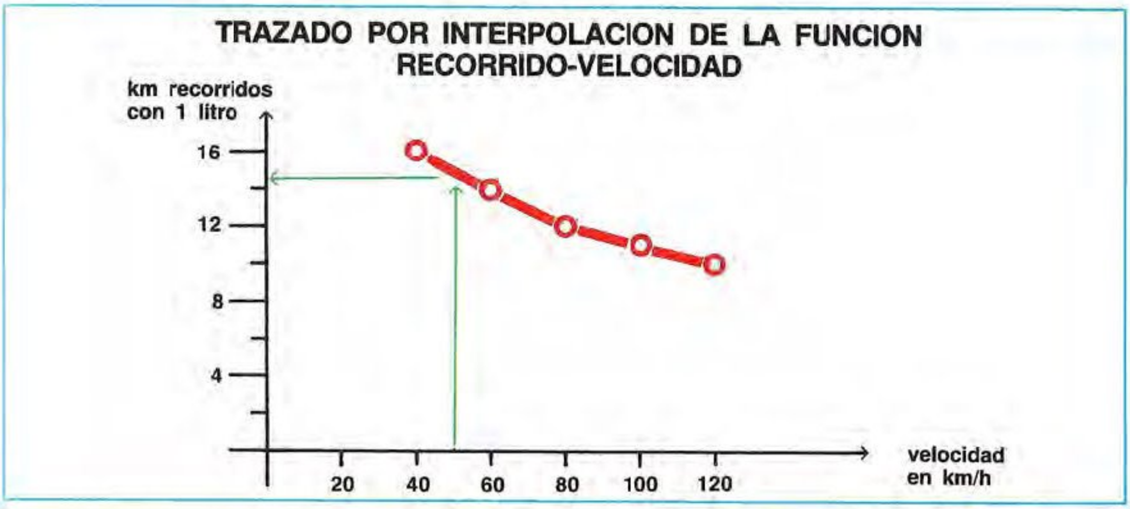
Los números $\frac{5}{2}$ y $\frac{1}{20}$, que multiplicados por los valores de las variables dan las longitudes correspondientes, se denominan **factores de escala.** En general, tenemos:

$$\left(\begin{array}{c}\textbf{longitud}\\ \textbf{en el eje}\end{array}\right) = \left(\begin{array}{c}\textbf{factor}\\ \textbf{de escala}\end{array}\right) \times \textbf{valor}$$

La tabla y, por tanto el gráfico, se elaboran a partir de los resultados de la observación de un fenómeno físico: cómo varía la distancia recorrida al variar la velocidad. Para caracterizar esta variación hemos elegido un cierto número de velocidades arbitrarias (40, 60, 80... km/h) y hemos medido los respectivos recorridos. Hasta el momento hemos obtenido una serie de puntos que reflejan las observaciones efectuadas y no la ley general de dependencia del recorrido con respecto a la velocidad. Sin embargo, si nuestras observaciones han tenido lugar para intervalos de velocidad no demasiado amplios, podemos pensar que el recorrido relativo a una velocidad intermedia entre dos de las experimentadas será a su vez intermedio entre los recorridos correspondientes a dichas velocidades.

En consecuencia, para obtener los puntos no medidos directamente, podemos unir los puntos conocidos con segmentos rectilíneos y suponer que sobre estos segmentos caen las parejas de valores x e y obtenibles experimentalmente. Así, hemos trazado el diagrama (ver pág. 437) de la función que liga el recorrido a la velocidad: a partir de dicho diagrama podemos obtener el valor de la distancia recorrida correspondiente a un valor cualquiera de la velocidad. En este diagrama se muestra, por ejemplo, el procedimiento que permite determinar el recorrido correspondiente a la velocidad de 50 km/h. En realidad, la función así obtenida es sólo aproximativa: la correspondencia exacta vendrá dada por una ley matemática que suministre, para cada valor de x, un valor de y. El diagrama de la función analítica exacta pasará, eso sí, por los puntos que hemos obtenido experimentalmente.

La definición de una función que ligue dos magnitudes x e y, cualesquiera que sean, nos permite establecer el valor de y asociado a un determinado valor de x. La magnitud cuyo valor puede establecerse arbitrariamente (en este caso x, la velocidad) se llama **variable independiente**; la otra magnitud (en este caso, la distancia recorrida), cuyo valor depende del de la variable in-



dependiente, es la **variable dependiente**. Normalmente, la variable independiente se representa sobre el eje x (abscisas) y la dependiente sobre el eje y (ordenadas).

El hecho de que la variable dependiente (recorrido, eje y) tenga un valor ligado al de la variable independiente (velocidad, eje x) se puede sintetizar en la forma simbólica **y = f(x),** que literalmente significa: «la variable dependiente y toma valores que son función de los tomados por la variable independiente x».

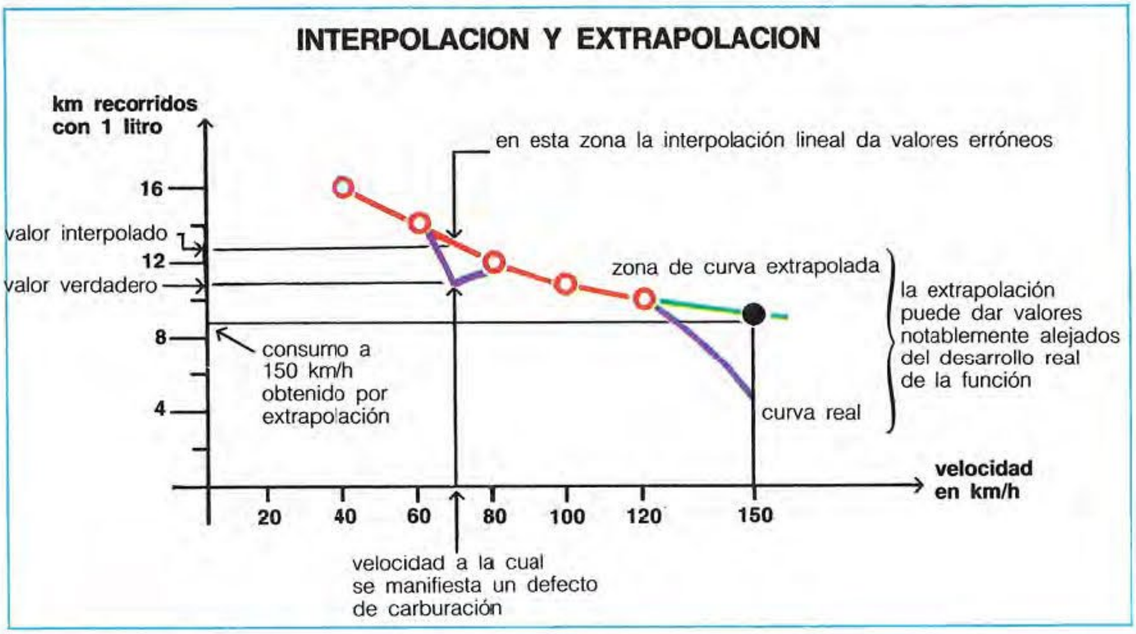
La representación gráfica de la función y = f(x) –recorrido = f(velocidad)– puede obtenerse eligiendo valores cualesquiera de la variable independiente (x, velocidad) y midiendo los correspondientes valores de la variable dependiente (y, recorrido). Esta libertad de elección de los valores de x no es absoluta; por ejemplo, no podríamos medir el recorrido para 200 km/h, puesto que el automóvil no llega a dicha velocidad. Para la elección de los valores de x tenemos, pues, que limitarnos a los valores posibles. El campo dentro del cual puede variar x se denomina **campo de definición** de la función; en este caso, el campo de definición está comprendido entre 10 km/h (para velocidades más bajas no tiene sentido el consumo) y la velocidad máxima del automóvil.

Interpolación y extrapolación

Al dibujar la curva del diagrama disponemos de un número limitado de puntos (40, 60, 80, 100, 120 km/h: cinco en total). Las partes comprendidas entre dos puntos han sido dibujadas uniéndolas con un segmento rectilíneo; esta operación que, intuitivamente, nos parece «natural» constituye, en realidad, un complejo artificio matemático: la **interpolación lineal**. Interpolar significa obtener a partir de dos o más valores los puntos intermedios que faltan, es decir, precisamente lo que hacemos al unir dos puntos del gráfico: el segmento que los une representa todos los puntos intermedios que faltan.

De este modo podemos leer también valores que no han sido medidos directamente: uniendo los puntos con velocidades de 40 y 60 km/h, hemos interpolado entre estas dos velocidades, obteniendo un segmento que representa todas las correspondencias intermedias y que permite, por ejemplo, obtener la distancia recorrida para 50 km/h. La interpolación lineal es un procedimiento casi siempre lícito desde el punto de vista matemático, pero sólo suministra valores muy próximos a los reales cuando no se dan variaciones bruscas en el desarrollo de la función. Por el contrario, si el gráfico evoluciona de forma compleja, los resultados suministrados por la interpolación pueden ser muy distintos de los reales. En efecto, supongamos que el vehículo con el que estamos efectuando las pruebas tenga un defecto de carburación a causa del cual, para velocidades comprendidas entre 60 y 80 km/h, el consumo aumenta, para luego normalizarse tras los 80 km/h (ver pág. 438). En este caso, a causa de la «punta» de consumo entre los 60 y los 80 la función recorrido/velocidad adopta un desarrollo complejo.

Si intentáramos obtener por interpolación lineal el recorrido asociado a la velocidad de 70 km/h, cometeríamos un error inadmisible. Una forma de evitar este riesgo consiste en efectuar otras pruebas para velocidades intermedias respecto



a las ya experimentadas, para así poder interpolar un mayor número de puntos.

A pesar de sus riesgos encubiertos, la interpolación es un procedimiento utilizado muy a menudo en los algoritmos científicos. Hay distintos métodos de cálculo por interpolación: además de la interpolación lineal, en la que los puntos conocidos se unen mediante segmentos rectilíneos, existe la posibilidad de usar líneas curvas, lo que a veces permite mejorar notablemente la precisión de los valores interpolados.

Otro artificio matemático muy utilizado es la **extrapolación**.

En las pruebas de las que hemos sacado la tabla de consumos, la velocidad máxima alcanzada es de 120 km/h. Supongamos que la velocidad máxima del automóvil sea 150 km/h (el campo de definición de la función recorrido/velocidad va de 10 a 150 km/h), pero que no se hayan hecho pruebas a esta velocidad. ¿Cómo podemos obtener los consumos entre 120 (último punto medido) y 150 km/h? Un método (extrapolación) consiste en «prolongar» la función hasta la velocidad de 150 km/h. También para la extrapolación hay métodos de cálculo matemáticos; pero así como para la interpolación nos limitamos a «inventar» una trayectoria entre dos puntos que conocemos (por lo que tenemos bastantes probabilidades de aproximarnos a la trayectoria real), en el caso de la extrapolación no sabemos nada sobre la configuración de la curva más allá del último punto medido. En

tales circunstancias es fácil suponer una configuración no real y obtener resultados erróneos. En el caso de la curva de consumos, por ejemplo, para velocidades próximas a la máxima del vehículo el consumo aumenta rápidamente, y la configuración de la curva varía bruscamente. No es, por tanto, posible una extrapolación basada en la configuración general para velocidades inferiores (ver diagrama de la pág. 438).

Representación analítica de las funciones

La representación gráfica de las funciones es útil para su comprensión y para observar su marcha general, pero no puede constituir un método de cálculo.

En el ejemplo del consumo del automóvil, si quisiéramos conocer el gasto de combustible para las distintas velocidades, tendríamos que usar el gráfico para obtener el consumo y luego multiplicar por el precio unitario del combustible.

Esta operación no puede efectuarse mediante ordenador, ya que la máquina no dispone de medios para leer un gráfico (al menos los modelos normales). El camino a seguir es la traducción del gráfico a una expresión matemática que indique cuáles son los cálculos a efectuar para obtener la distancia recorrida en función de la velocidad. En este caso concreto, una buena aproximación viene dada por la fórmula:

$$\left(\begin{array}{c}\text{km recorridos}\\ \text{con 1 litro}\end{array}\right) = 18.6 - 0.07 \times \left(\begin{array}{c}\text{velocidad}\\ \text{en km/h}\end{array}\right)$$

Utilizándola podemos conocer el recorrido para cada velocidad sustituyendo el valor de la misma y efectuando las operaciones indicadas. La expresión anterior es la **representación analítica** de la función.

En la programación, todas las funciones que se van a utilizar han de ser expresadas en forma analítica. Existen, a este respecto, sólo dos casos reales. En el primero, la función es conocida a priori y no hace falta más que traducir a la simbología adecuada las operaciones indicadas en la función misma. En el segundo, sólo se conocen algunos puntos –por ejemplo, medidos experimentalmente– y hay que obtener a partir de ellos la relación matemática (es el caso de la curva de consumos, que lleva a la ecuación que acabamos de ver). Hay diversos métodos, más o menos complejos, en función de la precisión que se desee obtener; el más simple, como hemos visto, es la interpolación lineal, o «regresión lineal», que supone una marcha rectilínea de la función entre los puntos conocidos.

El concepto de función puede aplicarse a cualquier ley que ligue entre sí dos (o más) magnitudes. La simbología genérica es siempre $y = f(x)$ (x e y son dos símbolos cualesquiera y pueden reemplazarse por otras letras).

En la programación existen funciones «intrínsecas» del lenguaje. Cada lenguaje tiene las suyas con una simbología propia, pero casi todos conservan la misma estructura formal:

$$\begin{array}{c}\text{variable}\\ \text{independiente}\end{array} = \text{NOMBRE}\left(\begin{array}{c}\text{variable}\\ \text{dependiente}\end{array}\right)$$

donde NOMBRE es una sigla alfabética que identifica a la función concreta que normalmente está formada por las primeras letras del nombre de la función.

Las posibilidades gráficas de los modernos ordenadores personales permiten representar eficazmente incluso funciones tridimensionales. En la fotografía, un histograma.



Olivetti

Las funciones en el Basic

El Basic es uno de los lenguajes más ricos en funciones ya preparadas. Su notable variedad, relativa sobre todo a las funciones dedicadas al tratamiento de las cadenas, facilita la tarea del programador a condición de que las conozca bien. Por lo tanto, se recomienda una atenta lectura de este capítulo. Las funciones del BASIC pueden dividirse en tres tipos:

- funciones numéricas
- funciones para el tratamiento de las cadenas
- funciones especiales (funciones I/O y de memoria)

Algunas funciones matemáticas (y sus posibles aplicaciones) implican ciertas nociones de trigonometría y de logaritmos. Puesto que son temas estrictamente ligados a la programación científica, las correspondientes instrucciones se presentan sólo a nivel informativo. Los lectores que deseen profundizar en el tema pueden remitirse a los numerosos textos de análisis matemático y trigonometría dedicados a ello.

Funciones numéricas

A este grupo pertenecen todas las funciones que suministran un valor numérico tras haber elaborado (con los adecuados algoritmos) un dato, siempre numérico, que se les suministró como parámetro.
Dicho dato se denomina **argumento** de la función. Para facilitar la lectura del párrafo, las funciones numéricas se presentan en orden creciente de dificultad. Las primeras son funciones utilizadas para efectuar cambios de clase de las variables, tales como redondeos, extracción de la parte entera, etc; siguen tres funciones matemáticas de uso corriente y, por último, las funciones trigonométricas y exponenciales.
Para cada grupo, el orden de presentación es el alfabético. El parámetro a suministrar (argumento) se indica con la letra N si se trata de un entero y con la letra R para los reales, teniendo en cuenta que tal distinción es sólo formal, puesto que si se usa un argumento de tipo no adecuado, el Basic mismo se encarga de la oportuna conversión.
Por último hay que recordar que, normalmente, el Basic interpretado no admite argumentos en doble precisión que, sin embargo, están previstos en el Basic compilado.

ABS(R). Suministra el valor absoluto (es decir con signo +) del argumento R. Como argumento puede utilizarse también una expresión. Por ejemplo:

```
Y = ABS(-10)
Z = ABS(3*(7-5)-2)
```

constituyen dos formas válidas de la función.
En el primer caso, el valor que se asigna a la variable Y es 10; en el segundo, Z = 4. Un ejemplo de aplicación de la función ABS(R) se tiene en las instrucciones condicionales con variable real.
En la comparación entre una variable real y el cero hay que utilizar el valor absoluto de dicha variable. Como vimos, el problema se resuelve comprobando el signo del parámetro y multiplicándolo por −1 si su signo era negativo (de este modo el valor final siempre será positivo); sin embargo, el mejor método consiste en usar la sentencia ABS(R).
Por ejemplo, si se desea determinar si la variable X tiene un valor (absoluto) menor o igual que 0.01, las instrucciones a introducir son:

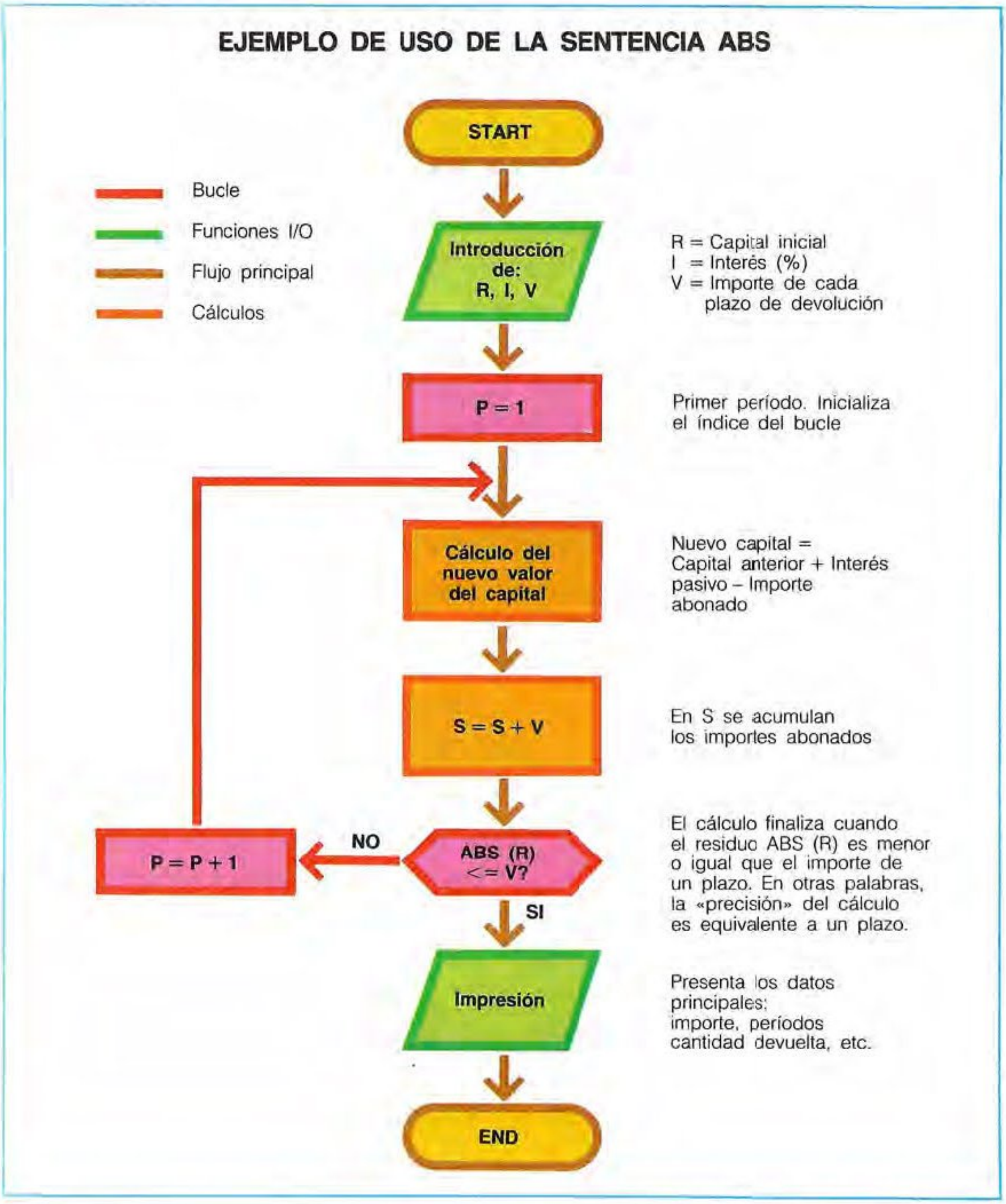
```
XI ABS(X)'XI contiene el valor absoluto de X IF
XI < = 0.01 THEN...
```

o bien, más concisamente:

```
IF ABS(X) < = 0.01 THEN...
```

Como ejemplo de aplicación, supongamos que se contrae una deuda de importe inicial R y con interés I a escalonar. Sea V el importe entregado en cada plazo; se desea calcular el número de plazos necesarios para la cancelación de la deuda, suponiendo que el cálculo del interés se efectúe al final de cada plazo.
El cálculo puede realizarse con las oportunas fórmulas de matemáticas financieras, pero puede también resolverse con un simple bucle de control sobre el valor del importe residual después de cada plazo. En efecto, al término de un plazo, al importe a restituir (R) ha de añadirse el interés pasivo (I*R/100) a lo que se ha de restar el importe entregado (V). Si el resultado es cero, la deuda se ha cancelado. De lo contrario, hay que considerar un nuevo período temporal. En la página contigua se muestra el correspondiente diagrama de flujo, mientras que en la pág. 442 aparece el programa.

EJEMPLO DE USO DE LA SENTENCIA ABS



CDBL(R). Convierte a R en un número en doble precisión. Por ejemplo:

```
10 R = 56.8
20 DR = CDBL(R)
```

En la variable DR se transfiere el mismo valor numérico contenido en R, pero en formato de doble precisión. Este formato reserva un mayor espacio de memoria para la variable y permite el desarrollo de cálculos con un mayor número de cifras significativas.

La sentencia CDBL(R) se utiliza antes de los cálculos que implican variables en simple y doble precisión, puesto que, al utilizar variables homogéneas, se eliminan posibles causas de error.

PROGRAMA PARA EL COMPUTO DEL INTERES

```
70 A$="££££££££££££"          'MASCARA DE IMPRESION (ESTA INSTRUCCION SE
80 '                           EXPLICARA MAS ADELANTE)
90 INPUT  "CAPITAL INICIAL"; R
100 C=R         'MEMORIZAR EN "C" EL VALOR DEL CAPITAL INICIAL "R"
110 INPUT "INTERES"; I
120 INPUT "IMPORTE DEL PLAZO"; V
140 P=1              ' PRIMER PERIODO. INICIALIZA EL INDICE DEL BUCLE
150 R=R+(I*R/100)-V
160 'LPRINT "R= ";R, "V= ";V:S$=INPUT$(1)
170 'NUEVO CAPITAL = CAPITAL ANTERIOR + INTERES PASIVO - IMPORTE ABONADO
190 S=S+V               'EN S SE ACUMULAN TODOS LOS IMPORTES ABONADOS
200 IF ABS (R) <=V GOTO 230 ELSE P=P+1: GOTO 150
210 'LA DEUDA SE CONSIDERA CANCELADA CUANDO EL CAPITAL RESIDUAL VALE 0.1
230 ' * INSTRUCCIONES DE IMPRESION *
240 '   *** ATENCION A LA SENTENCIA "LPRINT USING" SE EXPLICARA MAS ADELANTE ***
260 LPRINT "CALCULO DEL NUMERO DE PLAZOS NECESARIOS"
270 LPRINT "DE UN PRESTAMO CON INTERESES A ESCALONAR":LPRINT
280 LPRINT "CAPITAL INICIAL      R = ";: LPRINT USING A$;C
290 LPRINT "INTERESES            I = ";I
300 LPRINT "IMPORTE DEL PLAZO    V = ";: LPRINT USING A$;V
310 LPRINT
320 LPRINT "NUMERO DE PLAZOS     P = ";P
330 LPRINT "TOTAL ABONADO        S = ";: LPRINT USING A$;S
360 END
```

```
CALCULO DEL NUMERO DE PLAZOS NECESARIOS PARA CANCELAR
UN PRESTAMO CON INTERESES A ESCALONAR

CAPITAL INICIAL      R =      20000000
INTERESES            I =  10
IMPORTE DEL PLAZO    V =       3000000

NUMERO DE PLAZOS     P =  11
TOTAL ABONADO        S =      33000000
```

CINT(R). Convierte el real R en entero redondeando la parte decimal. Naturalmente, el resultado (entero) ha de estar comprendido en el intervalo – 32768/+ 32767 de validez de los enteros, de lo contrario se genera un error de desbordamiento, es decir, de superación de los límites. Por ejemplo:

A = CINT(7.51)
B = CINT(3574.2)
son instrucciones válidas

C = CINT(82754.3)
es un error: la parte entera es mayor que 32767

Esta instrucción puede ser muy útil para redondear los cálculos, a condición, sin embargo, de que el valor del entero no se salga de los límites previstos. Si esto ocurre, hacen falta métodos más complejos, que veremos más adelante.

CSNG(R). Es la función inversa de CDBL (R), y convierte el valor de R en precisión simple. Por ejemplo:

R = 135.7943 'constante en doble precisión
V = CSNG(R) 'asigna a V el valor de R, pero en
'simple precisión.

La función CSNG (R) se utiliza para reducir el espacio ocupado en la memoria o para homologar los factores que concurren en un cálculo.

FIX(R). Reduce el valor de R a su parte entera. Por ejemplo:

V = FIX (572.85) da V = 572
V = FIX (-374.6) da V = –374

Esta sentencia puede utilizarse, con algunos añadidos, para redondear los valores superiores al intervalo -32768/ + 32767 con que opera la función CINT.
Como puede verse en los dos ejemplos precedentes, la sentencia FIX (R) no tiene en cuenta

los valores decimales, por lo que su aplicación al número 572.85 da 572 y no 573, que sería el valor redondeado. Para obtener este último, hay que utilizar la sentencia FIX para sacar la parte entera del número y luego extraer (por diferencia) la parte decimal; finalmente, una prueba sobre el valor decimal suministra las indicaciones para el redondeo. Abajo se ve el diagrama de esta subrutina, y en la pág. 444 su listado. La subrutina sólo prevé el caso de números positivos, pero la generalización a valores negativos es inmediata. Téngase en cuenta que al redondear -374.6 se convierte en -373 y no en -375; por lo tanto hay que invertir la lógica.

INT(R). El valor suministrado es el mayor entero contenido en R. Por ejemplo:

N = INT (76.93) da N = 76

N = INT (–21.2) da N = –22, puesto que –22 es menor que –21.2
N = INT (–21.0) da N = –21

SNG(R). Suministra el signo del parámetro R codificado como 1, 0, – 1. Por ejemplo:

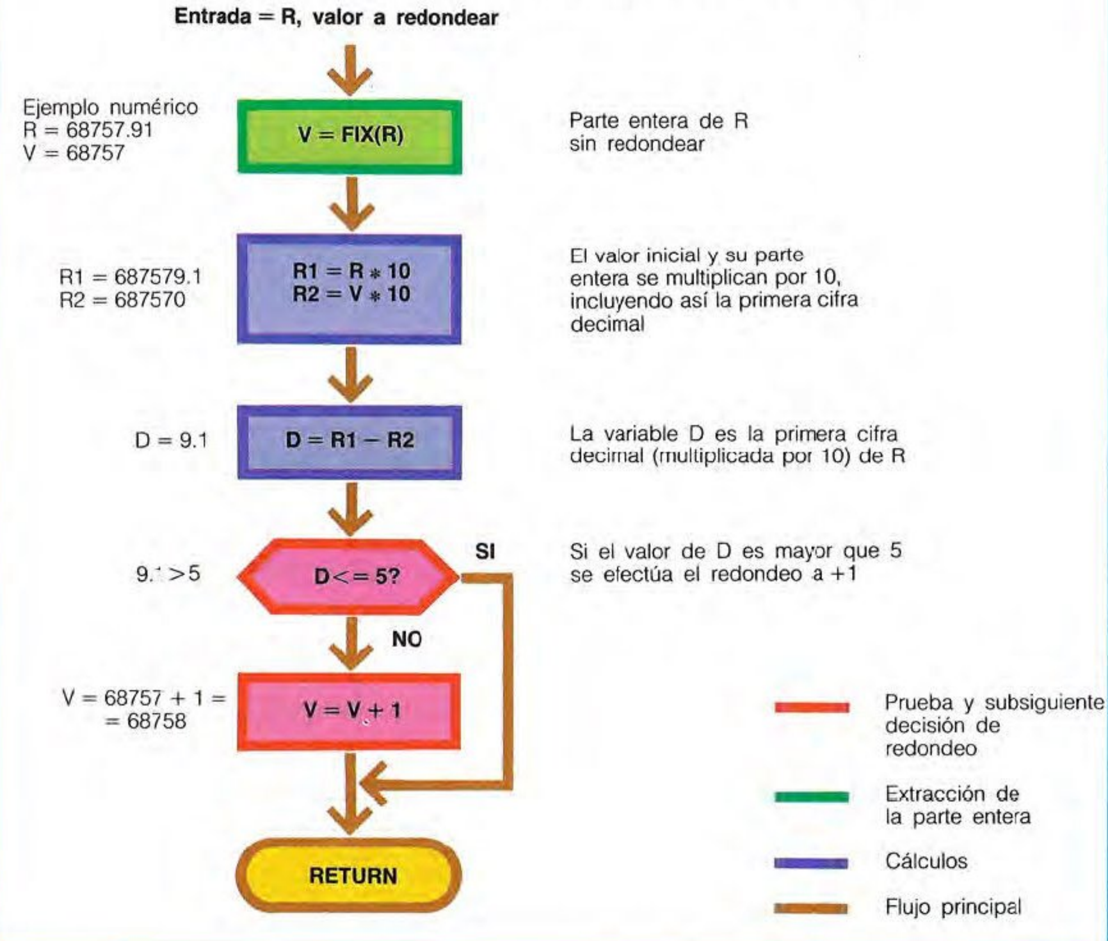
R>0 da el valor 1: Y = SNG (28) da Y = 1
R = 0 da 0: Y = SNG (0) da Y = 0
R<0 da –1: Y = SNG (–75) da Y = –1

SQR(R). Suministra el valor de la raíz cuadrada de R. Por ejemplo:

Y = SQR (16) da Y = 4
Y = SQR (15) da Y = 3.87298

El número R ha de ser positivo o cero: en el campo de los números reales no existen las raí-

DIAGRAMA DE LA SUBRUTINA DE REDONDEO



PROGRAMA PARA EL REDONDEO DE NUMEROS

```
15 A$="££££££££££.£"
20 INPUT "VALOR QUE SE DESEA REDONDEAR";R
30 '
40 GOSUB 1000
50 '
60 LPRINT "VALOR A REDONDEAR R = ";: LPRINT USING A$;R
70 LPRINT "VALOR REDONDEADO  V = ";: LPRINT USING A$;V
80 LPRINT
90 GOTO 20
100 END
1000 ' * SUBRUTINA DE REDONDEO *
1010 V=FIX (R)          PARTE ENTERA DE "R" SIN REDONDEAR
1020 R1=R*10
1030 R2=V*10
1040 '
1050 D=R1-R2            EN LA VARIABLE "D" ESTA LA PRIMERA CIFRA DECIMAL DE "R"
1060 IF D<=5 THEN RETURN ELSE V=V+1: RETURN


VALOR A REDONDEAR R=          123.4
VALOR REDONDEADO  V=          123.0

VALOR A REDONDEAR R=          123.5
VALOR REDONDEADO  V=          123.0
```

PROGRAMA PARA LA GENERACION DE NUMEROS ALEATORIOS

```
50 BI$=CHR$(27)+"+"+CHR$(7)      LIMPIA EL VIDEO Y EMITE UN SONIDO
60 PRINT BI$
70 PRINT "***.PROGRAMA PARA LA GENERACION DE NUMEROS ALEATORIOS ***"
80 PRINT
90 PRINT "         1) = EL GENERADOR DE NUMEROS ALEATORIOS HA DE"
100 PRINT "              SER PUESTO A CERO CON UN VALOR INTERNO"
110 PRINT
120 PRINT "        2) = EL GENERADOR DE NUMEROS ALEATORIOS HA DE"
130 PRINT "              SER PUESTO A CERO CON UN VALOR DADO POR EL OPERADOR"
140 PRINT
150 INPUT "INTRODUCIR EL NUMERO SELECCIONADO      " RESP
160 IF RESP=1 THEN RANDOMIZE(3): GOTO 200
180 PRINT BI$
190 RANDOMIZE
200 INPUT "CUANTOS VALORES SE DESEA IMPRIMIR";N
210 IF N=0 THEN STOP
215 LPRINT "PROGRAMA PARA LA GENERACION DE NUMEROS ALEATORIOS"
217 LPRINT
220 I=1
230 Y=RND
240 LPRINT Y
250 IF I=N THEN LPRINT: GOTO 60
260 I=I+1
270 GOTO 230
280 END

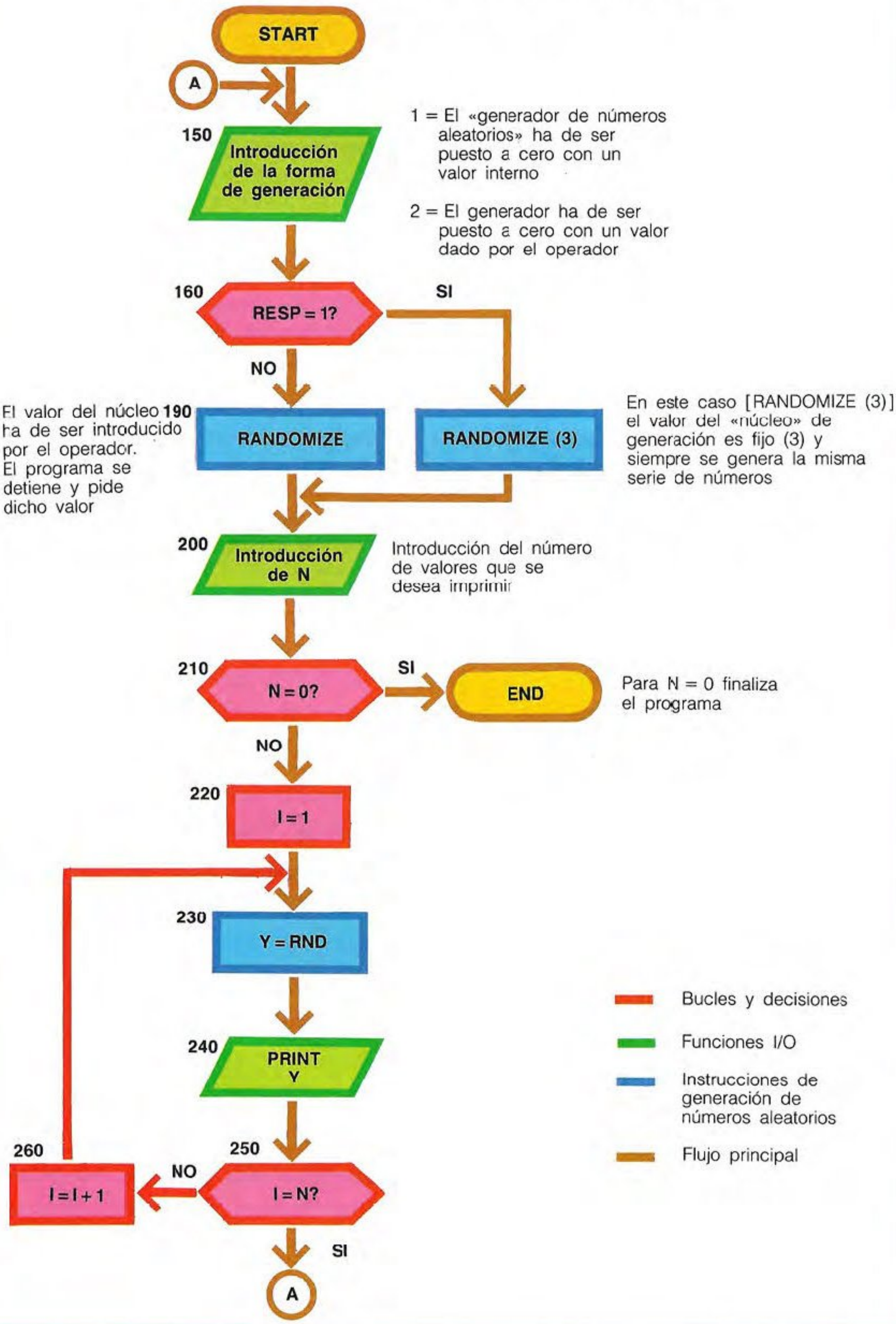

PROGRAMA PARA LA GENERACION DE NUMEROS ALEATORIOS

     .88598
     .484668
     .586328
```

ces cuadradas o de índice par de los números negativos.

RND(R). Genera un número aleatorio (random) comprendido entre 0 y 1. El concepto de número aleatorio es bastante complejo y, para una definición más detallada, se remite al lector que esté interesado a los textos de estadística. A

GENERACION DE NUMEROS ALEATORIOS



efectos de programación, la función RND (R) se puede entender como un algoritmo que, aplicado repetidamente, suministra una serie de números tomados al azar y distintos entre sí, o mejor dicho, con escasas probabilidades de que haya dos iguales.
La generación de números aleatorios se realiza en dos fases consecutivas. En la primera se predispone el "generador de números aleatorios" con la sentencia RANDOMIZE (N), en la cual N es un valor entero cualquiera.
Esta sentencia indica a la máquina el valor (N) a utilizar como argumento para la sucesiva generación, con la sentencia RND (R), de una serie de números aleatorios.
Si se omite RANDOMIZE, se generan siempre los mismos números, puesto que el núcleo sobre el que se construye la serie no varía.
En la segunda fase, con la sentencia RND, se activa el procedimiento de generación.
La sentencia RND puede utilizarse con argumento, es decir, en la forma RND (R), o sin él. Al usar la forma con argumento, si $R \neq 0$ se genera la misma secuencia para cualquier valor de R, mientras que para $R = 0$ se repite la última secuencia generada.
También la sentencia RANDOMIZE puede utilizarse con o sin el parámetro N. Si se omite el parámetro, el ordenador detiene la ejecución y pide su valor al usuario.
En la pág. 444 se muestra un programa para la generación de números aleatorios, mientras que en la pág. 445 se puede observar el diagrama correspondiente.

TEST 13

1 / Una línea de programa realiza el cálculo A = B/(C-D). ¿En qué caso se puede generar un error capaz de detener el programa?

2 / ¿De qué forma se puede prevenir el error de la pregunta anterior?

3 / Algunas de las siguientes sentencias son erróneas. ¿Cuáles?

a) IF K = 2.5 THEN 300
b) IF L + M = 6 GOTO 150
c) IF L = 5 RETURN
d) A = – 5:B = 2:K = A – B:ON K GOTO 10,20,30

4 / Escribir un programa para comprobar si el valor de la variable V está comprendido entre 2.6 y 15.8 (exclusive e inclusive).

5 / ¿Qué valores toman las variables E y F en el siguiente programa?

```
10 A = – 18.75
20 B = 20
30 C = FIX(A)
40 D = INT(A)
50 E = B + C
60 F = B + D
```

6 / Escribir las instrucciones para el control del signo de la variable V. La salida ha de ser un escrito que diga si el valor de la variable es mayor, igual o menor que 0.

7 / ¿Qué instrucción hay que añadir al siguiente programa para no caer en error?

```
10 DEFINT A – Z
20 B = – 9
30 C = SQR(B)
```

Las soluciones, en la pág. 451

ATN(Y). En esta función el argumento (real) está indicado con la letra Y en lugar de la R para no crear confusión con la función TAN, que expondremos más adelante.
Suministra el valor del arco trigonométrico cuya tangente vale Y (función **arcotangente**).
El arco calculado por la función ATN se expresa en radianes; por lo tanto, el resultado está comprendido en el intervalo $-\pi/2 + \pi/2$.
El símbolo π indica la relación entre la longitud de la circunferencia de un círculo cualquiera y su diámetro: π = circunferencia/diámetro; su valor es 3.1415926...
El valor de Y puede estar también en doble precisión, mientras que el resultado suministrado (el arco) está casi siempre en simple precisión.

COS(R). Calcula el coseno de R, que ha de estar expresado en radianes (función **coseno**). También esta función está normalmente en simple precisión.
Recuérdese que la medida de los ángulos se expresa normalmente en grados sexagesimales; para obtener el correspondiente valor en radianes hay que recurrir a la proporción:

$$180:\pi = \text{Angulo en grados} : \text{Angulo en radianes}$$

$$\text{Angulo en radianes} = \frac{\pi \times \text{Angulo en grados}}{180} =$$

$$= 0.01745 \times \text{Angulo en grados}$$

SIN(R). Calcula el seno de R (función **seno**); también para esta función R ha de estar expresado en radianes.

TAN(R). Calcula el valor de la tangente de R, también expresado en radianes (función **tangente**). Obsérvese que las funciones TAN(R) y ATN(Y) son la una el inverso de la otra: TAN(R) suministra la tangente del ángulo R, mientras ATN(Y) suministra el ángulo cuya tangente es R; mediante fórmulas:

Y = TAN(R)
Y es el valor de la tangente del ángulo R
R = ATN(Y)
R es el valor del ángulo cuya tangente es Y

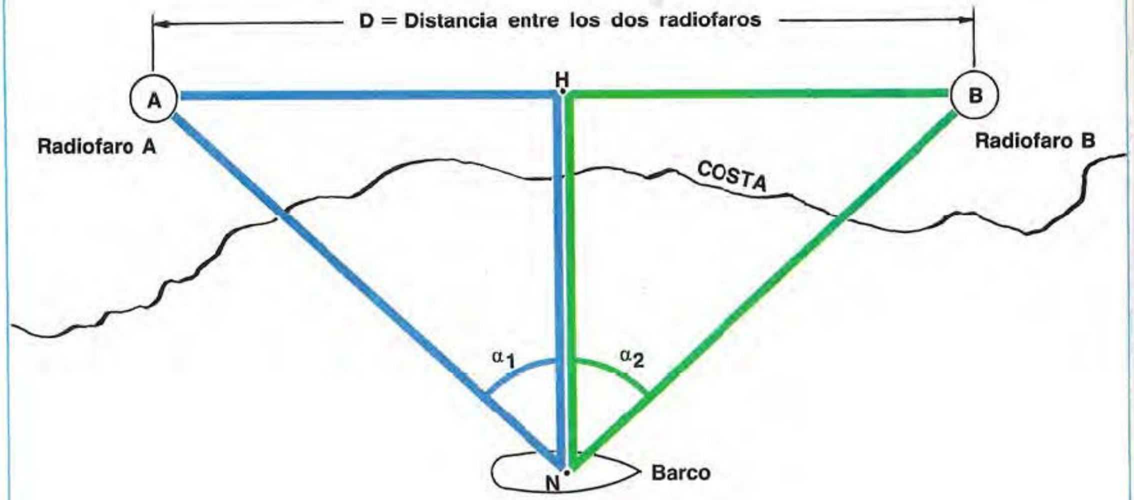
En la tabla de abajo se muestran las listas y las salidas de un programa que ilustra el uso de las funciones SIN(R), COS(R) y ATN(R).

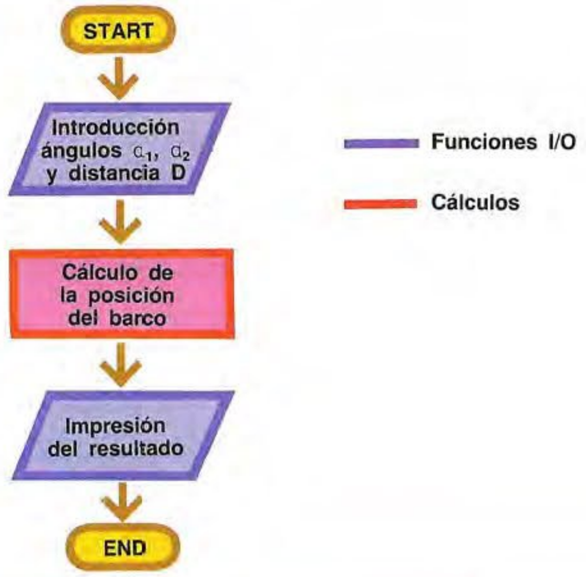
EJEMPLO DE USO DE LAS FUNCIONES TRIGONOMETRICAS

```
40 ' RG = ANGULO EXPRESADO EN GRADOS SEXAGESIMALES
50 '
60 ' RR = VALOR CORRESPONDIENTE EN RADIANES
70 '
100 INPUT "ANGULO (EN GRADOS SEXAGESIMALES)";RG
110 '
120 IF RG=99 GOTO 1000
130 '
140 ' * TRANSFORMACION EN RADIANES *
150 '
160 RR=.01745*RG
170 '
180 '
190 ' * FUNCIONES TRIGONOMETRICAS
200 '
210 S=SIN(RR)              ' SENO
220 C=COS(RR)              ' COSENO
225 T=S/C                  ' TANGENTE = SENO/COSENO
230 A=ATN(T)               ' ARCOTANGENTE, HA DE CORRESPONDER A (RR)
240 '
250 ' * IMPRESION *
260 '
270 LPRINT "ANGULO EN GRADOS SEXAGES. = ";RG, "CORRESPONDIENTE EN RADIANES = ";RR
280 LPRINT "SENO = ";S,"COSENO = ";C, "ARCOTANGENTE = ";A
285 LPRINT
290 GOTO 100
300 '
1000 END
```

```
ANGULO EN GRADOS SEXAGESIMALES =  20       CORRESPONDIENTE EN RADIANES =  .349
SENO =  .341958                COSENO = .939715              ARCOTANGENTE =  .349
```

APLICACION DE LAS FUNCIONES TRIGONOMETRICAS A UN PROBLEMA DE TRIANGULACION





Aplicaciones de las funciones trigonométricas. Las funciones trigonométricas (seno, coseno, tangente, arcotangente) son indispensables en los problemas de «triangulación», es decir, en todos los casos en que hay que obtener distancias o alturas de puntos geográficos con medidas de ángulos. Un ejemplo de aplicación es el cálculo de la posición de un barco, basado en la detección de señales de radio emitidas por dos radiofaros.

En la aplicación real, el cálculo viene simplificado por el hecho de que todas las posiciones (radiofaros y embarcación) se expresan en términos de longitud y latitud, según el sistema de referencias geodésico basado en los meridianos y paralelos. En este ejemplo, a título indicativo, omitiremos este aspecto del problema para obtener, simplemente, la distancia de la embarcación a la costa. Los radiofaros transmiten señales de radio que son recibidas a bordo de la nave por un receptor especial (radiogoniómetro) capaz de suministrar los ángulos (α_1 y α_2 en el

gráfico contiguo) con los cuales llegan las señales. Midiendo estos dos ángulos y conociendo la distancia D entre los radiofaros, se puede calcular la distancia de la nave a la costa (el segmento NH en el dibujo) realizando algunos cálculos sencillos con los triángulos rectángulos NHA (color azul) y NHB (color verde).
La fórmula de resolución es:

$$NH = \frac{\text{distancia}}{\text{de la costa}} = \frac{\text{distancia entre los radiofaros}}{tg(\alpha_1) + tg(\alpha_2)}$$

Los símbolos tg (α_1) y tg (α_2) representan las tangentes de los ángulos α_1 y α_2. La función trigonométrica tangente, aunque está explícitamente prevista en el Basic, puede obtenerse a partir de las funciones seno y coseno con la fórmula:

$$tg(R) = \frac{SIN(R)}{COS(R)}$$

Por lo tanto, el programa tendrá que realizar las siguientes funciones:

1 / Introducción de los ángulos α_1 y α_2 y de la distancia D
2 / Transformación de los ángulos en radianes
3 / Cálculo de la tangente de cada ángulo
4 / Cálculo de la distancia a la costa
5 / Presentación del resultado

Abajo se muestra el listado de este programa. Este ejemplo sirve especialmente para mostrar los métodos a usar en el caso de que hagan falta funciones trigonométricas no previstas en Basic. Dicho lenguaje posee sólo las funciones seno, coseno y arcotangente; todas las demás han de ser planteadas por el programador.
El tema será ampliado al final de este capítulo.

EXP(R). Calcula el valor del número *e* elevado a la potencia R (el número *e* = 2.7183 es la base de los logaritmos neperianos). En otras palabras, la sentencia

Y = EXP(R)

da el valor $Y = e^R$. Por ejemplo, con R = 2 tenemos:

Y = EXP(2), $Y = e^2 = 2.7183^2 = 7.389$...

También esta función, de uso estrictamente

CALCULO DE LA DISTANCIA DE UN BARCO A LA COSTA

```
30 PRINT "N.B. LOS ANGULOS HAN DE SER INTRODUCIDOS EN GRADOS SEXAGESIMALES
40 '
50 INPUT "ALFA 1";I
55 A=I            ' MEMORIZAR EN "A" EL VALOR ORIGINAL DE ALFA 1
60 INPUT "ALFA 2";II
65 B=II           ' MEMORIZAR EN "B" EL VALOR ORIGINAL DE ALFA 2
70 INPUT "DISTANCIA ENTRE LOS RADIOFAROS";D
80 ' TRANSFORMACION DE LOS GRADOS SEXAGESIMALES EN RADIANES
90 I=I*.01745
100 II=II*.01745
110 '
120 'CALCULO DE LA DISTANCIA NH = D / TAN (I) + TAN (II)
130 NH=D/(TAN (I)+TAN(II))
140 '
150 '
160 ' * INSTRUCCIONES DE IMPRESION *
170 LPRINT "DISTANCIA ENTRE LOS DOS RADIOFAROS" ;D
180 LPRINT "ANGULO ALFA 1 = ";A
190 LPRINT "ANGULO ALFA 2 = ";B
200 LPRINT "N.B. LOS ANGULOS ESTAN EXPRESADOS EN GRADOS SEXAGESIMALES
210 LPRINT                        ' ESPACIADO
220 LPRINT "DISTANCIA DEL BARCO A LA COSTA NH = ";NH
230 LPRINT:LPRINT                 ' DOBLE ESPACIADO
240 GOTO 50
250 END


DISTANCIA ENTRE LOS DOS RADIOFAROS 1455
ANGULO ALFA 1 = 12.1555
ANGULO ALFA 2 = 39.4546
N.B. LOS ANGULOS ESTAN EXPRESADOS EN GRADOS SEXAGESIMALES

DISTANCIA DEL BARCO A LA COSTA    NH = 1401.54
```

científico, se explica con mayor detalle en los textos de análisis matemático.

LOG(R). Calcula el valor del logaritmo natural (neperiano) del número R.
No nos adentraremos en la definición del término logaritmo. Baste recordar que R ha de ser mayor que cero.

Funciones de cadena

La peculiaridad del lenguaje Basic se manifiesta especialmente en su amplio repertorio de funciones para el tratamiento de las cadenas de caracteres. El uso de estas funciones permite una notable flexibilidad y facilita la escritura de programas para la elaboración de archivos o la preparación de tablas. La variedad de funciones para el tratamiento de las cadenas y su forma específica dependen de la versión concreta de Basic utilizada. En este capítulo se describen las funciones más comunes y específicas del Basic 80; más adelante el tema se complementará con la descripción de algunas estructuras que prevén el uso de subcadenas y de algunas funciones no incluidas en la forma estándar, sino específicas de una versión concreta del Basic. Téngase en cuenta que en algunas formas del Basic los datos no pueden memorizarse en disco sino en formato ASCII; por lo tanto, todos los valores numéricos, antes de ser escritos en disco, han de transformarse en caracteres; en la fase de lectura del disco, los mismos datos han de sufrir la transformación inversa: del código ASCII al numérico.
Estas técnicas se exponen detalladamente al final del capítulo.

ASC(A$). Suministra el valor numérico correspondiente al código ASCII del primer carácter de la cadena A$. Por ejemplo:

10 A$ = "17ABC"
(la cadena A$ contiene los caracteres 1, 7, A, B, C)
20 X = ASC(A$)
(se extrae el código correspondiente al primer carácter)
30 PRINT X
    49
(se imprime el valor de X)

En el ejemplo, la función X = ASC(A$) restituye el valor numérico 49, que es el código ASCII del carácter 1 (todos los símbolos numéricos contenidos en una cadena están codificados como caracteres). En la tabla inferior se muestra un programa que ejemplifica esta sentencia.

CHR$(N). Da una cadena de un solo elemento constituido por el símbolo correspondiente al código N (en la codificación ASCII).
Por ejemplo:

10 N = 65
20 B$ = CHR$(N)
30 PRINT B$
    A

El código 65 corresponde a la letra A. En la pág. 452 se muestra un programa para la conversión en carácter del valor numérico N introducido mediante teclado. En el programa se incluye el control de que N esté comprendido entre los códigos correspondientes a las letras y los números. Además, se puede activar o no la reescritu-

EJEMPLO DE USO DE LA SENTENCIA ASC (A$)

```
10 ' ** EJEMPLO DE USO DE LA SENTENCIA ASC(A$) **
15 '                FILE = ISTASC
20 DEFINT A-Z                 'TODAS LAS VARIABLES SE DEFINEN COMO ENTERAS
30 INPUT "INSERTAR UNA CADENA CUALQUIERA                  ",A$
35 IF A$="FIN" GOTO 100         'EL PROGRAMA TERMINA INTRODUCIENDO LA PALABRA FIN
50 N=ASC(A$)                'SE EXTRAE EL CODIGO DEL PRIMER ELEMENTO
60 LPRINT A$,N              'IMPRESION DE LA CADENA Y DEL CODIGO
80 LPRINT                   'INSTRUCCION DE ESPACIADO EN LA IMPRESION
90 GOTO 30
100 END

ABCDEFGT        65

BCFGGHYK        66

COMO            67

A               65
```

SOLUCIONES DEL TEST 13

1-2 / (Las respuestas de las preguntas 1 y 2 se han reunido).
El error se produce cuando C=D. En este caso, la diferencia C-D se hace 0, y no se puede dividir por 0. El error puede eliminarse saltándose el cálculo bajo la condición C=D. Esta parte puede escribirse así:

```
210 IF C=D GOTO 2000
220 A=B/(C-D)
225 ' * Punto de reentrada *
... otras instrucciones del programa ...
2000' * * Error * *
2010 PRINT "Error: división por cero"
2020 INPUT "Introducir 1 para continuar"; N
2030 IF N=1 GOTO 225
2040 STOP
```

Al verificarse la condición C=D, el programa salta a la línea 2000, escribe el diagnóstico (línea 2010) y se pregunta si ha de proseguir (línea 2020). En caso de respuesta afirmativa (N=1, línea 2030) vuelve a realizar las instrucciones que siguen a la 220 (retorno a la línea 225). La 220 no es ejecutada y el programa puede continuar.

---

3 / Las instrucciones erróneas son la c) y la d).
La línea c) es una instrucción de RETURN condicional. El condicionamiento de una instrucción se obtiene siempre con IF... THEN..., excepto para la sentencia GOTO. La forma correcta es, por tanto: IF L=5 THEN RETURN.
La línea d) utiliza el valor K para direccionar los saltos a las instrucciones 10, 20 y 30. En este tipo de instrucción el valor de la variable no puede ser menor que cero. Además la línea d) no tiene sentido desde el punto de vista lógico, pues K se obtiene por diferencia entre dos valores constantes (A y B) y es también un valor constante.

---

4 / El control consiste en comprobar que V sea simultáneamente (AND) mayor que 2.6 y menor que 15.8: IF 2.6 <V AND V<15.8 THEN PRINT "Comprendido"
Aquí los extremos (2.6 y 15.8) no están comprendidos. Para incluirlos, la expresión sería: IF 2.6<=V AND V<=15.8 THEN PRINT "Comprendido"

---

5 / La sentencia C=FIX(A) reduce el valor de A a su parte entera; tenemos, por tanto, C=-18. La D=INT(A) extrae el mayor entero contenido en A; tenemos, pues, D=-19 (cuidado con el signo: el valor -18 no está contenido en -18.75).
Por lo tanto: E=20-18=2 F=20-19=1

---

6 / La sentencia a usar es SNG(V). El programa puede escribirse en la forma:

```
10 K=SNG(V)
20 IF K=-1 THEN PRINT "Menor que cero"
30 IF K=0 THEN PRINT "Igual a cero"
40 IF K=1 THEN PRINT "Mayor que cero"
```

---

7 / El valor de SQR(B) no existe (en el campo de los números reales) para valores negativos del parámetro B. Por tanto, la instrucción a añadir es el cambio de signo de la variable B: 25 IF B<0 THEN B = –1 * B
La instrucción es condicional, para que no cambie el signo si B es positivo.

PROGRAMA PARA EL USO DE LA SENTENCIA CHR$

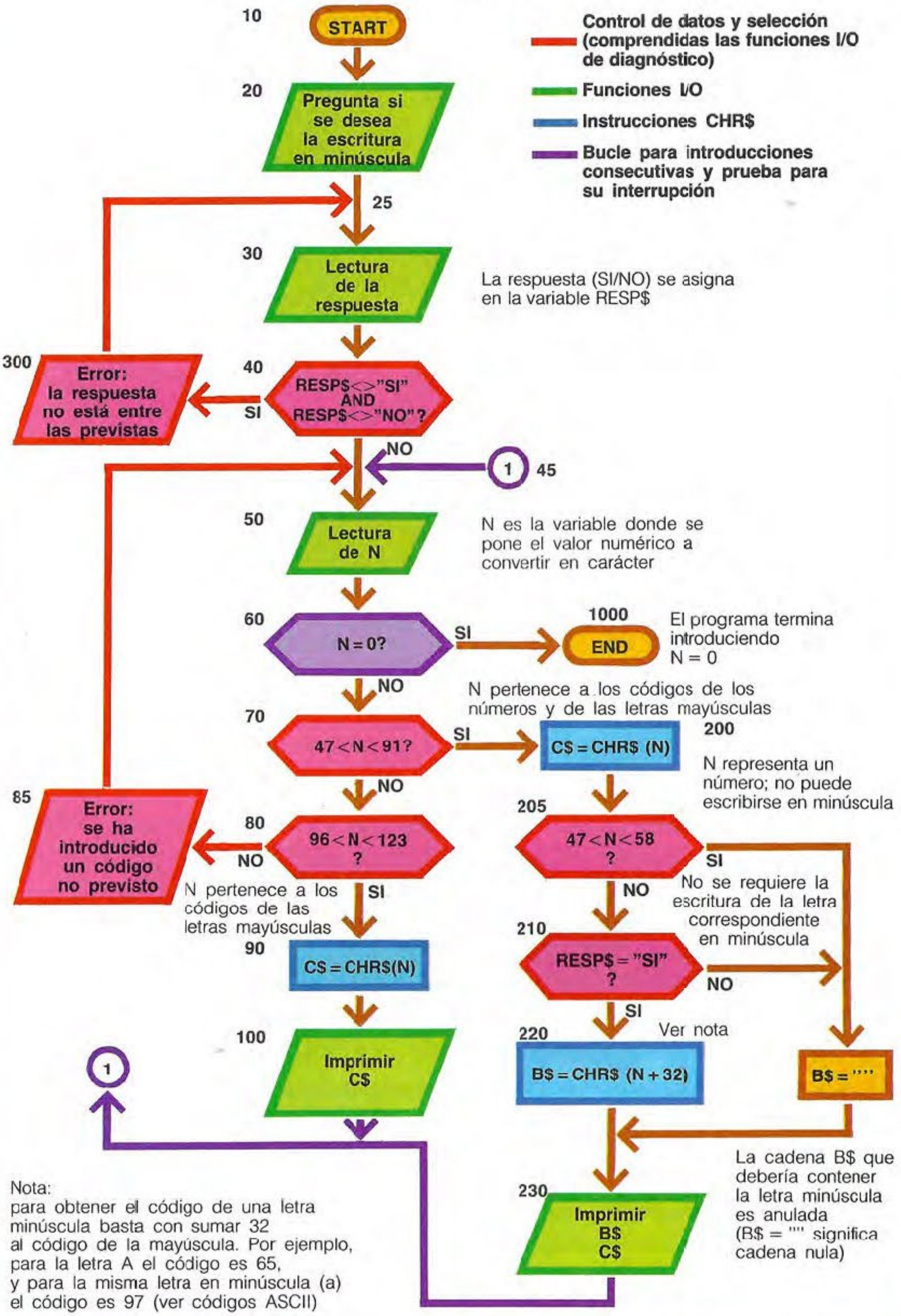
```
10 ' ** PROGRAMA PARA EL USO DE LA SENTENCIA CHR$  **
15 '              FILE = ISTCHR
17 BI$=CHR$(27)+"+"+CHR$(7)
18 PRINT BI$
20 PRINT "SE DESEA UN DUPLICADO DE ESCRITURA EN MINUSCULA (SI/NO)"
25 '
30 INPUT RESP$
40 IF RESP$<>"SI" AND RESP$<>"NO" THEN GOTO 300
45 ' COMIENZO DE BUCLE
46 PRINT BI$
47 PRINT "INTRODUCIR (0) PARA TERMINAR"
50 INPUT "VALOR NUMERICO A CONVERTIR EN CARACTER";N
60 IF N=0 GOTO 1000
70 IF 47<N AND  N<91 GOTO 200
80 IF 96<N AND N<123 GOTO 90
85 PRINT "ERROR, HA SIDO INTRODUCIDO UN CODIGO NO PREVISTO": GOTO 50
90 CS=CHR$(N)
100 LPRINT "VALOR NUMERICO";N,C$
110 GOTO 45
200 CS=CHR$(N)
205 IF 47<N AND N<58 THEN B$="" :GOTO 230
210 IF RESP$="SI" THEN GOTO 220 ELSE B$="" :GOTO 230
220 BS= CHR$(N+32)
230 LPRINT "VALOR NUMERICO";N.B$.C$
240 GOTO 45
300 PRINT "ERROR, LA RESPUESTA NO ESTA ENTRE LAS PREVISTAS"
301 GOTO 25
1000 END
```

```
VALOR NUMERICO 65          a          A
VALOR NUMERICO 66          b          B
VALOR NUMERICO 67          c          C
VALOR NUMERICO 68          d          D
```

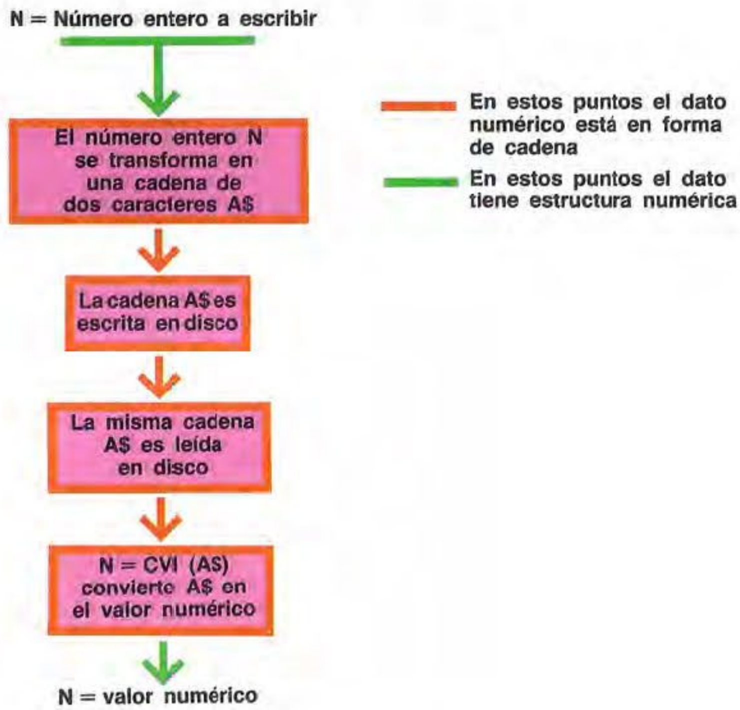
ra, en minúscula, de la letra introducida (véase el capítulo de los códigos ASCII). El diagrama correspondiente aparece en la pág. contigua. La sentencia CHR$(N) se usa principalmente para enviar los códigos especiales (transparentes) al vídeo y a la impresora. Por ejemplo, poniendo A$ = CHR$(12) y escribiendo la cadena A$ en la impresora se obtiene un salto de página (el código 12 corresponde a FF, salto de página). El tema se ampliará más adelante.

CVI(A$). Convierte una cadena de dos caracteres en su equivalente valor numérico entero (dos bytes). La sentencia CVI(A$) se usa principalmente para leer en disco; en efecto, los datos, incluso los numéricos, son memorizados en forma de cadena, y así son leídos. En el programa hay que prever las oportunas instrucciones de reconversión en valor numérico antes de utilizar para un cálculo cualquiera los datos leídos en el disco. En la pág. 454 se muestra el diagrama que describe cualitativamente el procedimiento de escritura y lectura de valores numéricos en el disco.

USO DE LA SENTENCIA CHR$



ESQUEMA LOGICO DE ESCRITURA Y LECTURA EN DISCO DE UN VALOR NUMERICO (ENTERO)



CVS(A$). Convierte una cadena de cuatro caracteres (cuatro bytes) en su equivalente numérico real en precisión simple.

CVD(A$). Convierte una cadena de ocho caracteres en un real en doble precisión.

El uso de las tres últimas funciones (CVI, CVS y CVD), junto con las funciones inversas que convierten los números en cadenas, se describe detalladamente en el capítulo dedicado a la memorización de los datos en disco.
Al utilizar estas sentencias hay que tener en cuenta que la representación de un número en forma de cadena ocupa mucho más espacio que su representación binaria.
Por ejemplo, el número 12745 es un entero y puede representarse, como número binario, utilizando sólo dos bytes (o sea 16 bits: 0 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1), mientras que en la representación ASCII cada cifra es un carácter y, como tal, ocupa un byte; por lo tanto, para representar el número 12745 en código ASCII hacen falta cinco bytes (la cadena que representa el 12745 es: 49, 50, 55, 52, 53; ver tablas ASCII).

HEX$(N). Convierte un entero N en una cadena cuyos caracteres son la representación hexadecimal del argumento (N). Por ejemplo:

```
10 DEFINT N
20 N = 21
30 E$ = HEX$(N)
```

transfiere a la cadena E$ la representación hexadecimal del número 21 (decimal).
En la página contigua se muestran la lista y algunas salidas de un programa que ilustra el uso de la sentencia HEX$(N).
De por sí, la función opera sólo con números enteros, pero puede aplicarse también a argumentos reales, en cuyo caso el sistema mismo se encarga de convertir el real en entero (este mecanismo es común a todas las funciones que requieren un argumento entero).

INKEY$. Permite leer una cadena de un solo ca-

rácter introducido mediante el teclado. Es una de las funciones utilizadas en las operaciones I/O, y será descrita con más detalle en el capítulo relativo a dicho tema.
La sintaxis de esta sentencia, llamando C$ a una cadena genérica, es la siguiente:

C$ = INKEY$

Tras esta sentencia, el sistema se pone a la espera de un carácter que habrá que introducir mediante el teclado; al recibirlo, lo transferirá a la cadena C$.

INPUT$(N). Es análoga a la anterior, y ofrece, además, la posibilidad de establecer cuántos caracteres han de ser leídos poniendo el parámetro N igual a dicho número. Por ejemplo:

C$ = INPUT$(5)

acepta cinco caracteres del teclado y los transfiere a la cadena genérica C$.
También para esta función, los ejemplos aplicativos se darán en el capítulo dedicado a las operaciones de I/O.

INSTR(N,A$,B$). Permite determinar si la cadena B$ está contenida en la A$.
Como respuesta, suministra un valor numérico (comprendido entre 1 y 255) que indica en qué posición se da la correspondencia. Si B$ no está contenida en A$, la respuesta es 0. El parámetro N, que puede omitirse, indica (si lo hay) en qué posición de A$ ha de comenzar la búsqueda. Por ejemplo, las instrucciones en la forma más simple (sin parámetro N):

```
10 A$ = "NOMBRE NO PRESENTE"
20 B$ = "E"
30 K = INSTR(A$,B$)
```

dan K = 6, pues la cadena B$ (formada por el único carácter E) ha sido hallada en la posición 6 de la cadena A$ (es el 6.° carácter de A$).
Análogamente, poniendo B$ = "S", la sentencia K = INSTR(A$,B$) daría K = 14, puesto que la letra S está en la posición 14 de la cadena A$. (los espacios en blanco también cuentan como caracteres).
En la segunda forma –INSTR(N,A$,B$)– se puede especificar con el parámetro N en qué posición ha de empezar la búsqueda.
Por ejemplo, con B$ = "E", en el ejemplo anterior, la sentencia:

K = INSTR(7,A$,B$)

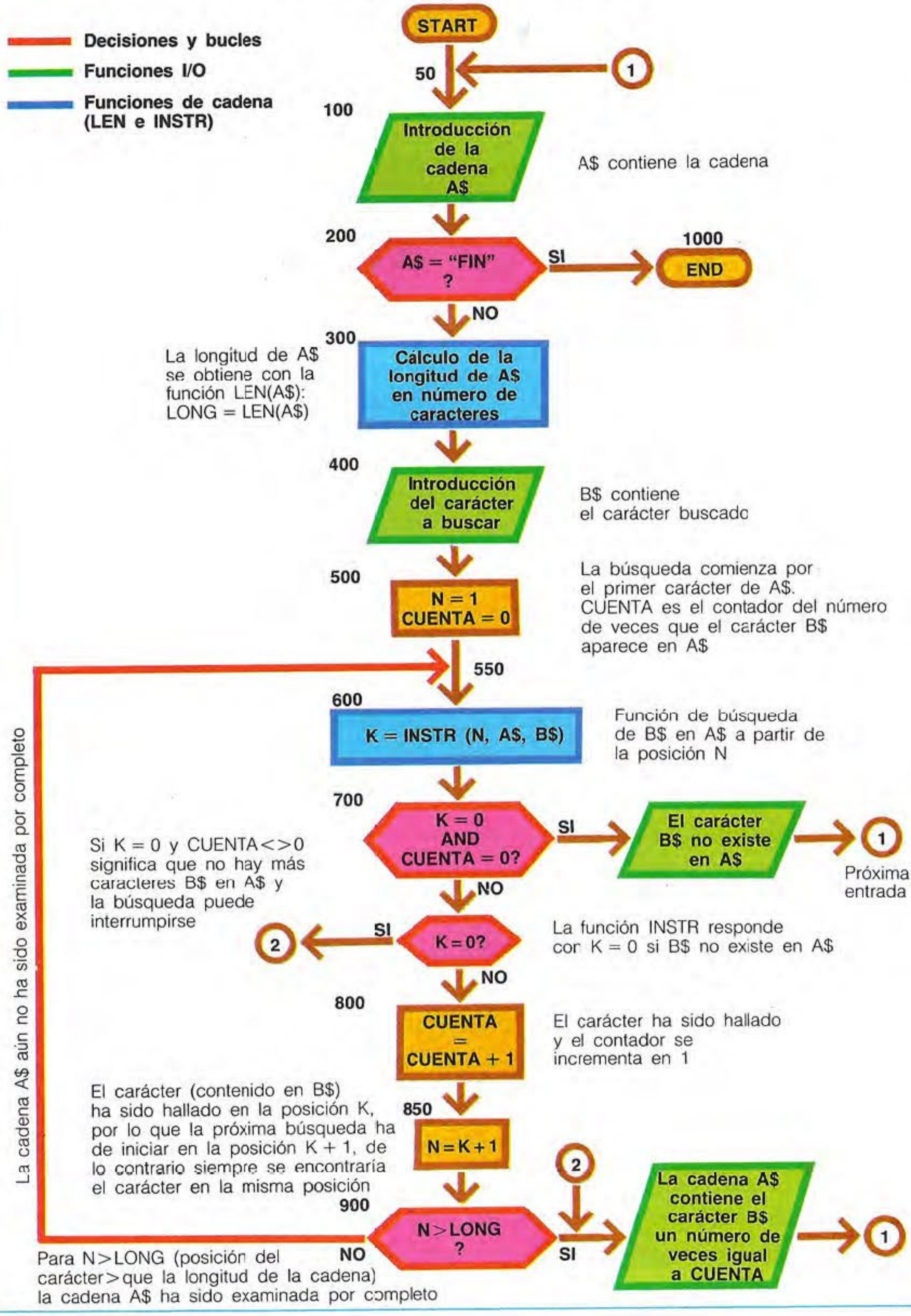
daría K =13, puesto que la búsqueda comienza en el carácter número 7 de A$, y la primera letra E (la de NOMBRE) no se cuenta.
En la pág. 456 se muestra el diagrama de flujo de un programa que lee cualquier cadena de

USO DE LA FUNCION HEX$(N)

```
30 DEFINT N
40 BI$=CHR$(27)+"+ +CHR$(7)       'LIMPIA EL VIDEO Y EMITE UN "BEEP"
50 '     PRIMER CASO: ARGUMENTO ENTERO
60 PRINT BI$
70 PRINT"INTRODUCIENDO (0) TERMINA EL PRIMER CASO"
80 INPUT "ARGUMENTO;N                  'N es declarado entero en la línea 30
90 IF N=0 GOTO 130
100 A$=HEX$(N)
110 LPRINT "ARGUMENTO  : ";N,"VALOR HEXADECIMAL        ";A$
120 GOTO 80
130 LPRINT
140 '     SEGUNDO CASO  : ARGUMENTO REAL
145 '   N.B. LA FUNCION HEX$(N) CONVIERTE LOS ARGUMENTOS SIEMPRE EN ENTEROS
150 PRINT BI$
160 PRINT"INTRODUCIENDO (0) TERMINA EL PROGRAMA"
170 INPUT "ARGUMENTO";R                  'R es una variable real
180 IF R=0 GOTO 220
190 A$=HEX$(R)               'CONVIERTE (R) EN HEXADECIMAL Y LO MEMORIZA EN A$
200 LPRINT "ARGUMENTO  :   ";R,"VALOR HEXADECIMAL          ";A$
210 GOTO 170
220 END

ARGUMENTO :    123              VALOR HEXADECIMAL        7B
ARGUMENTO :    456              VALOR HEXADECIMAL        1C8
ARGUMENTO :    789              VALOR HEXADECIMAL        315
```

DIAGRAMA PARA EL CALCULO DE LAS RECURRENCIAS DE UN CARACTER EN UNA CADENA



caracteres introducidos mediante teclado, pregunta qué carácter hay que buscar y suministra, en salida, el número de veces que dicho carácter aparece en la cadena introducida.
En esta página se muestra el correspondiente listado. En el programa se utiliza la función LEN(A$), que expondremos más adelante.

LEFT$(A$,N). Extrae de la cadena A$ los N primeros caracteres de la izquierda. Por ejemplo, el programa:

10 A$ = "VELÁZQUEZ"
20 B$ = LEFT$(A$,4)

da B$ = VELA, puesto que se han pedido los cuatro primeros caracteres (empezando por la izquierda) de la cadena A$.
El valor de N ha de estar entre 1 y 255; poniendo N = 0, la cadena B$ queda anulada.

En las págs. 458 y 459 se muestra el diagrama de un programa que utiliza la instrucción LEFT$ para la gestión de un archivo de direcciones. Los datos contenidos en el archivo son:

Campo 1: Apellido y Nombre
(30 caracteres, los 20 primeros para el apellido)
Campo 2: Ciudad, Calle, Número
(40 caracteres, los 10 primeros para la ciudad)

El archivo reside en disco y contiene mil direcciones; se desea la impresión de todos los nombres de una determinada ciudad, o bien de todas las direcciones de las personas con el mismo apellido (cualquiera que sea la ciudad).
En este caso, el diagrama ha de prever dos posibilidades:

CALCULO DE LAS «RECURRENCIAS» DE UN CARACTER EN UNA CADENA

```
10 ' **PROGRAMA PARA EL CALCULO DE LAS "RECURRENCIAS" DE UN CARACTER **
20 '    EN UNA CADENA
30 '
40 '        FILE = RICO
50 '
90 PRINT "PARA SALIR DEL PROGRAMA INTRODUCIR (FIN)"
100 INPUT "INSERTAR UNA CADENA   ",A$
200 IF A$="FIN" GOTO 1000
300 LONG=LEN(A$)
310 ' La línea (300) memoriza en la variable (LONG) el número de
320 ' caracteres contenidos en la cadena (A$)
400 INPUT "INTRODUCIR EL CARACTER A BUSCAR      ",B$
500 N=1
510 CUENTA=0      CUENTA es el contador del número de veces que el
520 '             carácter B$ aparece en A$
550 '
600 K=INSTR(N,A$,B$)
700 IF K=0 AND CUENTA=0 THEN GOTO 910
702 IF K=0 GOTO 902
800 CUENTA=CUENTA+1
850 N=K+1
900 IF N<LONG GOTO 550
902 LPRINT "EL CARACTER";" (";B$;") ","ESTA CONTENIDO EN LA CADENA"
903 LPRINT " (";A$;") ";"UN NUMERO DE VECES IGUAL A:";CUENTA
904 LPRINT
905 GOTO 50
910 LPRINT "EL CARACTER ";" (";B$;")" "NO ESTA CONTENIDO EN LA CADENA"
920 LPRINT "(";A$;")": GOTO 50
1000 END
```

```
EL CARACTER (N) ESTA CONTENIDO EN LA CADENA
(ANTONIO) UN NUMERO DE VECES IGUAL A: 2

EL CARACTER (A) ESTA CONTENIDO EN LA CADENA
(CLAUDIA) UN NUMERO DE VECES IGUAL A: 2

EL CARACTER (E) ESTA CONTENIDO EN LA CADENA
(EMILIO) UN NUMERO DE VECES IGUAL A: 1

EL CARACTER (K) NO ESTA CONTENIDO EN LA CADENA (JAIME)
```

1 / selección por ciudades (10 primeros caracteres del campo 2)
2 / selección por apellidos (20 primeros caracteres del campo 1)

En la pág. 460 se muestra parte del listado (se omite la parte relativa a la lectura de los datos del disco).

LEN(A$). Da la longitud de la cadena A$ expresada en número de caracteres, incluyendo en la cuenta los espacios en blanco o eventuales caracteres especiales no imprimibles. Un ejemplo del uso de esta sentencia se da en el gráfico de la pág. 456 y en el listado de la pág. 457.

MID$(A$,NS,NC). Extrae de la cadena A$ un número de caracteres igual a NC, a partir de la posición NS. Por ejemplo, el programa:

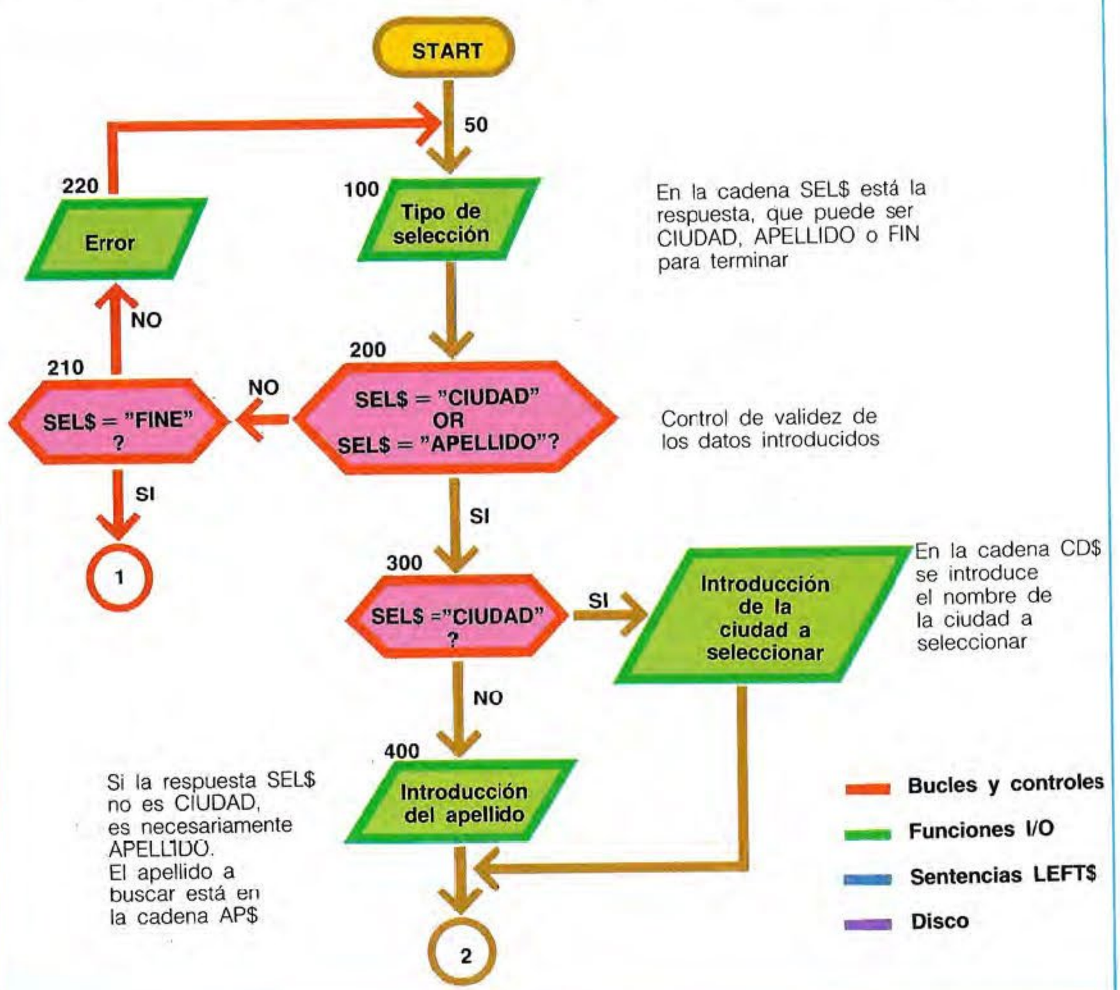
```
10 A$ = "ESTA ES UNA CADENA DE PRUEBA"
20 B$ = MID$(A$,4,7)
```

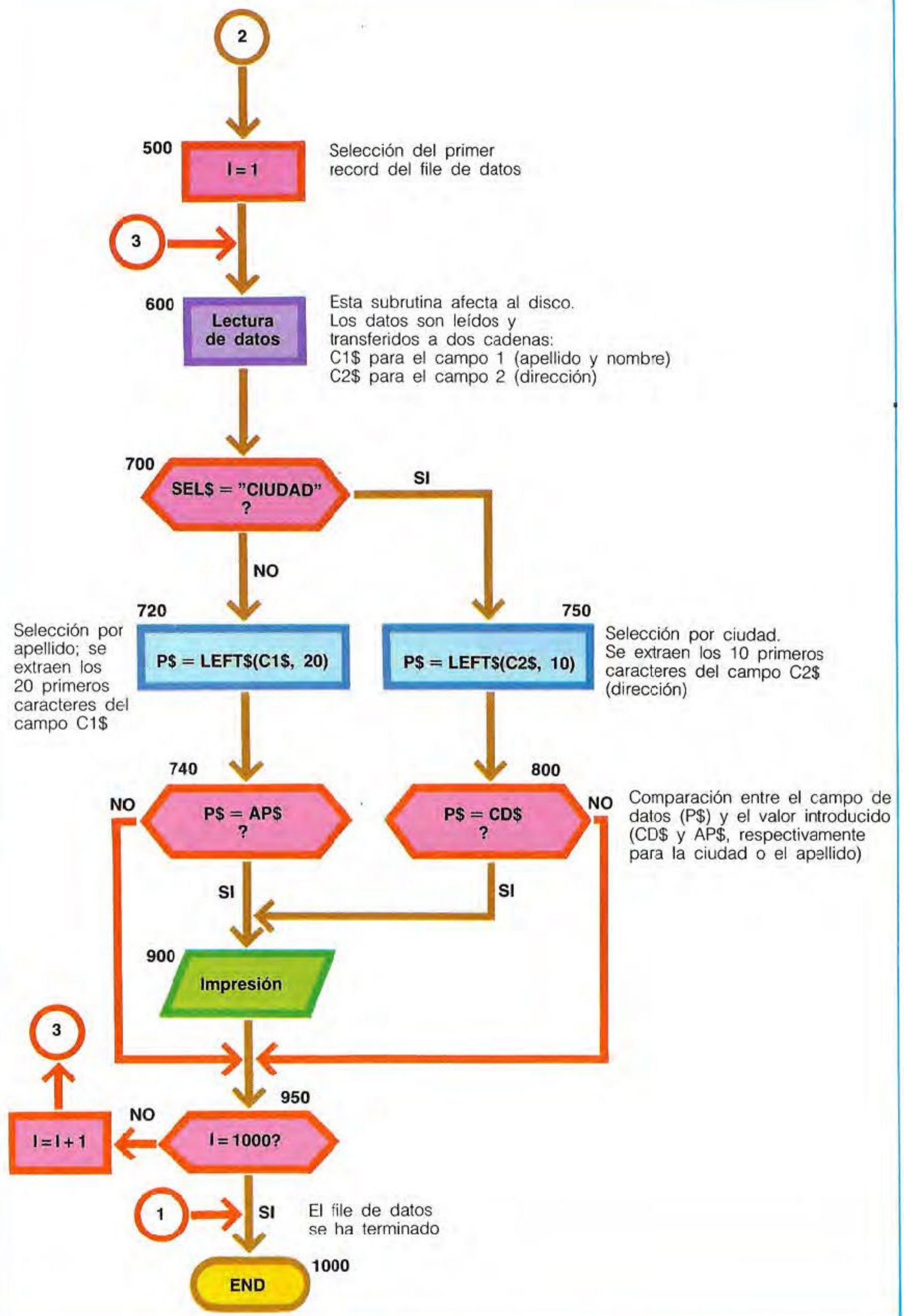
transfiere a B$ el contenido de A$ a partir del carácter número 4 (letra A de la palabra ESTA) y por un total de 7 caracteres (hasta la letra N de la palabra UNA; los espacios se cuentan como caracteres); por tanto tenemos: B$ = "A ES UN". La sentencia MID$ puede utilizarse también para efectuar una sustitución de partes de una cadena. La forma es:

MID$(A$,N,NC) = B$

El contenido de la cadena A$, a partir de la posición N y por una longitud de NC caracteres, es sustituido por el contenido de la cadena B$.

DIAGRAMA DE LA SELECCION DE DATOS DE UN ARCHIVO DE DIRECCIONES





2
500
I=1
Selección del primer
record del file de datos
3
600
Lectura
de datos
Esta subrutina afecta al disco.
Los datos son leídos y
transferidos a dos cadenas:
C1$ para el campo 1 (apellido y nombre)
C2$ para el campo 2 (dirección)
700
SEL$ = "CIUDAD"
?
SI
NO
720
750
Selección por
apellido; se
extraen los
20 primeros
caracteres del
campo C1$
P$ = LEFT$(C1$, 20)
P$ = LEFT$(C2$, 10)
Selección por ciudad.
Se extraen los 10 primeros
caracteres del campo C2$
(dirección)
740
800
NO
P$ = AP$
?
P$ = CD$
?
NO
Comparación entre el campo de
datos (P$) y el valor introducido
(CD$ y AP$, respectivamente
para la ciudad o el apellido)
SI
SI
900
Impresión
3
950
NO
I=I+1
I=1000?
1
SI
El file de datos
se ha terminado
1000
END

PROGRAMA PARA LA SELECCION DE DATOS DE UN ARCHIVO

```
10 ' ** PROGRAMA PARA LA SELECCION DE DATOS DE UN ARCHIVO **
20 '
30 '        FILE = SELDAT
50 '
100 PRINT "INTRODUCIR OPCION ELECCION"
105 PRINT
110 INPUT "TIPO DE SELECCION : CIUDAD, APELLIDO, FIN  ",SEL$
200 IF SEL$="CIUDAD" OR SEL$="APELLIDO" GOTO 300
210 IF SEL$= "FIN" GOTO 1000
220 PRINT "ERROR : TIPO DE SELECCION NO PREVISTA":GOTO 50
300 IF SEL$="CIUDAD" THEN INPUT "INTRODUCCION DE LA CIUDAD A SELECCIONAR  ",CD$
400 INPUT "INTRODUCCION DEL APELLIDO A SELECCIONAR  ",AP$
500 I=1      ' SELECCION DEL PRIMER REGISTRO DEL FILE DE DATOS
550 '
600 GOSUB 2000           ' LECTURA DATOS
700 IF SEL$="CIUDAD" GOTO 750
720 P$=LEFT$(C1$, 20) 'SE EXTRAEN LOS 20 PRIMEROS CARACTERES DEL CAMPO C1$
730 '                    QUE CONTIENE (APELLIDO Y NOMBRE)
740 IF P$=AP$ GOTO 900 ELSE GOTO 950
750 P$=LEFT$(C2$,10)  'SE EXTRAEN LOS 10 PRIMEROS CARACTERES DEL CAMPO C2$
760 '                    QUE CONTIENE (DIRECCION)
800 IF P$=CD$ GOTO 900 ELSE GOTO 950
900 ' INSTRUCCIONES DE IMPRESION
902 '
904 '
906 '
908 '
910 '
920 '
950 IF I=1000 GOTO 1000 ELSE I=I+1;GOTO 550
1000 END
2000 ' ** SUBRUTINA QUE AFECTA AL DISCO **
2010 '
2020 ' SE EXPONDRA MAS ADELANTE EN CAPITULO APARTE
2030 '
2040 RETURN
```

Por ejemplo, el programa:

10 B$ = "M"
20 A$ = "Gran Vía n. 84, B"
30 MID$(A$,17,1) = B$

da A$ = "Gran Vía n. 84, M". En la pág. 461 aparece la lista de un programa que ejemplifica el uso de las dos formas de esta sentencia.

MKI$(N). Convierte un entero (o una expresión entera) en una cadena de dos bytes; es la función inversa de CVI(A$).

MKS$(R). Convierte un número real R en una cadena de cuatro caracteres; es la función inversa de CVS.

MKD$(D). Convierte un real en doble precisión (D) en una cadena de ocho caracteres; es la función inversa de CVD.
El grupo de sentencias MKI$, MKS$ y MKD$ se usa sobre todo para transferir al disco valores numéricos; viceversa, en las operaciones de lectura se utilizan las funciones inversas (CVI, CVS, CVD). El tema se ampliará en el capítulo dedicado a la memoria masiva.

OCT$(N). Convierte un entero N en una cadena que contiene los caracteres que constituyen la representación octal de N. Tras la ejecución de las líneas

10 N = 16
20 A$ = OCT$(N)

la cadena A$ contiene los caracteres 20, que son el equivalente octal del decimal 16.
Esta sentencia y la HEX$ se han usado para generar las tablas de la pág. 63 a la 67. La lista del programa usado, para ello aparece en la pág. 462. Algunas de las sentencias usadas aún no han sido descritas, y saldrán más adelante.

RIGHT$(A$,N). Extrae N caracteres de la cadena A$, empezando por la derecha. Por ejemplo, las líneas

10 A$ = "MARIO PEREZ"
20 B$ = RIGHT$(A$,4)

transfieren a B$ cuatro caracteres de A$ empezando por la derecha; por tanto, tenemos: B$ = "EREZ".
Poniendo N = 0 [B$ = RIGHT$(A$,0)] se obtiene una cadena nula (B$ = ""), mientras que si N es igual a la longitud total de A$ se transfiere a B$ la cadena A$ completa.

En la pág. 463 se muestra el diagrama de una subrutina que inserta un carácter en una posición cualquiera de una cadena; la lista correspondiente está en la parte superior de la pág. 464, mientras que en las págs. 464 y 465 se muestra el diagrama de flujo de una rutina que borra un carácter cualquiera de una cadena (la

EJEMPLO DE USO DE LA SENTENCIA MID$

```
10 ' ** EJEMPLO DE USO DE LA SENTENCIA MID$ **
12 '
14 '        FILE = ESEIST
16 '
18 '     ASIGNACIONES
20 '                  A$ = Contiene la cadena examinada
22 '                  NS = Número del carácter donde comienza la extracción
24 '                  NC = Número de caracteres a extraer de la cadena A$
26 '
30 ' * PRIMERA FORMA      extracción de un grupo de caracteres de una cadena *
40 INPUT "CADENA              ";A$
50 INPUT "POR QUE CARACTER HA DE EMPEZAR LA EXTRACCION",NS
60 IF NS=0 GOTO 250       ' SI NS=0 TERMINA EL PROGRAMA
70 INPUT "CUANTOS SON LOS CARACTERES A EXTRAER",NC
80 IF NC<1 OR NC>255 THEN PRINT "ERROR":GOTO 70    'CONTROLES
90 K=NC+NS-1
100 IF LEN(A$)<K THEN LPRINT " LA CADENA ";A$; " ES DEMASIADO CORTA PARA"
101 IF LEN(A$)<K THEN LPRINT " PODER EXTRAER ";NC; "CARACTERES"
102 ' ***** ATENCION ***** en las líneas 90 y 100 hay un ALGORITMO
104 ' que controla si es posible extraer los caracteres NC a
106 ' partir de la posición NS
110 IF LEN(A$)<K GOTO 40
120 B$=MID$(A$, NS, NC)
130 LPRINT "CADENA INICIAL : ";A$
140 LPRINT "CARACTERES EXTRAIDOS : "B$
150 '
160 ' * SEGUNDA FORMA *
170 '
180 INPUT "INTRODUCIR LOS CARACTERES SUSTITUTIVOS ":C$
190 INPUT "EN QUE CARACTER HA DE EMPEZAR LA SUSTITUCION ":N
200 INPUT "PARA CUANTOS CARACTERES ":NC
210 IF LEN(C$)<NC THEN PRINT "ERROR":GOTO 180
220 MID$(A$, N, NC)=C$
222 LPRINT "CARACTERES SUSTITUTIVOS ":C$
230 LPRINT "CADENA CON CARACTERES SUSTITUIDOS ":A$
232 LPRINT
240 GOTO 40
250 END
```

```
CADENA INICIAL : AMALIA
CARACTERES EXTRAIDOS : A
CARACTERES SUSTITUTIVOS : E
CADENA CON CARACTERES SUSTITUIDOS : AMELIA

CADENA INICIAL : JULIA
CARACTERES EXTRAIDOS : A
CARACTERES SUSTITUTIVOS : O
CADENA CON CARACTERES SUSTITUIDOS : JULIO

CADENA INICIAL : SERGIO
CARACTERES EXTRAIDOS : ERG
CARACTERES SUSTITUTIVOS : ILV
CADENA CON CARACTERES SUSTITUIDOS : SILVIO
```

PROGRAMA DE CONVERSION

```
15 ' ** DECLARACION DE LAS VARIABLES Y DE LOS ARRAY
20 OPTION BASE 1
25 DEFINT A-P
30 DIM B(16)            'Array de los valores Binarios (16)
35 LPRINT TAB (2); "DECIMAL"; TAB (20); "BINARIO";TAB (50);"OCT";TAB(65);"HEX"
36 LPRINT:LPRINT
40 '
45 A1$=" Valor no admitido" ' Diagnóstico sobre los valores de entrada
100 ' ** ENTRADA DE VALORES Y CONTROLES
105 '
110 INPUT" Valor inicial";NSTAR
115 IF NSTAR<0 OR NSTAR>32000 THEN PRINT A1$:GOTO 110
120 INPUT" Valor final";NEND
125 IF NEND<NSTAR THEN PRINT A1$: GOTO 120
130 '
135 ' ** Comienzo del Bucle entre los valores NSTAR y NEND
140 '
145 FOR N=NSTAR TO NEND
150 '
155 ' ** Transformación en Binario
160 '
165 GOSUB 1000
170 '
175 ' ** Transformación en Octal
180 '
185 A3$=OCT$(N)
190 '
195 ' ** Transformación en Hexadecimal
200 '
205 A4$=HEX$(N)
210 '
215 ' ** IMPRESION
220 '
225 ' ** Transformar B(16) en Cadena
230 A2$=""
235 FOR I=16 TO 1 STEP -1
240 A2$=A2$+STR$(B(I))
245 NEXT I
275 LPRINT TAB(2);N;TAB(10);A2$;TAB(50);A3$;TAB(65);A4$
295 NEXT N
300 END

1000 '
1005 ' ** Convertir un entero en su representación Binaria
1010 ' ENTRADA : N= Número Entero a convertir
1015 ' SALIDA  : B(16)= Array con los símbolos 1,0 de la representación Binaria
1020 '
1025 '
1030 '
1035 FOR I=1 TO 16:B(I)=0:NEXT I   ' Puesta a cero
1040 M=N    ' Salvar el dato de Entrada para restituirlo invariado
1045 I=1   'I= Posición del resultado 1,0 en B(16)
1046 V=M/2     ' La división está en una variable real por tener resto
1050 K=M-INT(V)*2   ' K=1 si M no es divis. por 2;K=0 si M es divis.
1055 B(I)=K    ' B(i)=0/1
1060 IF M=1 GOTO 1090     'Fin?
1065 IF K=1 THEN M=M-1   ' Restar 1 si M no es divis.
1070 M=M/2
1075 I=I+1
1080 GOTO 1046
1105 RETURN
```

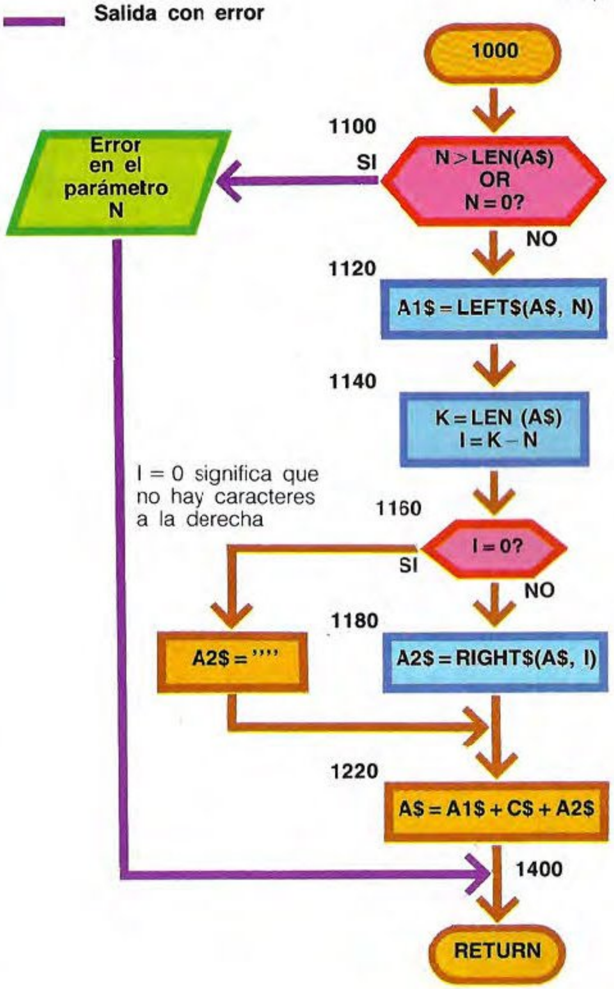
lista correspondiente está incluida en el programa que aparece en la pág. 466).

Ambos procedimientos ilustrados utilizan las funciones de cadena intrínsecas al Basic y constituyen un útil complemento, puesto que las funciones de inserción y cancelación de un carácter no están previstas en el repertorio estándar de funciones Basic.

SUBRUTINA DE INSERCION DE UN CARACTER EN UNA CADENA

- **Funciones I/O** (verde)
- **Decisiones** (rojo)
- **Funciones de cadena Basic** (azul)
- **Salida con error** (violeta)

Datos de entrada en la rutina:
A$ = Cadena a modificar
C$ = Carácter a insertar
N = Posición en número de caracteres empezando por la izquierda



Control de validez de los parámetros. El valor de N no puede ser mayor que la longitud de la cadena ni igual a 0

Extraer los caracteres de A$ que se hallan a la izquierda del punto en que hay que insertar el carácter C$

K = Longitud de la cadena
I = Número de caracteres a la derecha de la posición de inserción

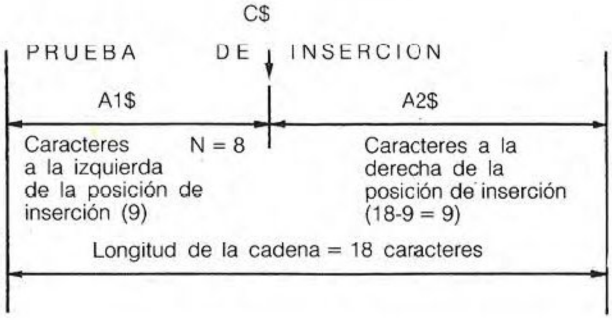
Extraer los caracteres de la derecha

Componer la cadena final

Ejemplo

Cadena en entrada = A$ =

Carácter a insertar:
C$ = espacio en blanco

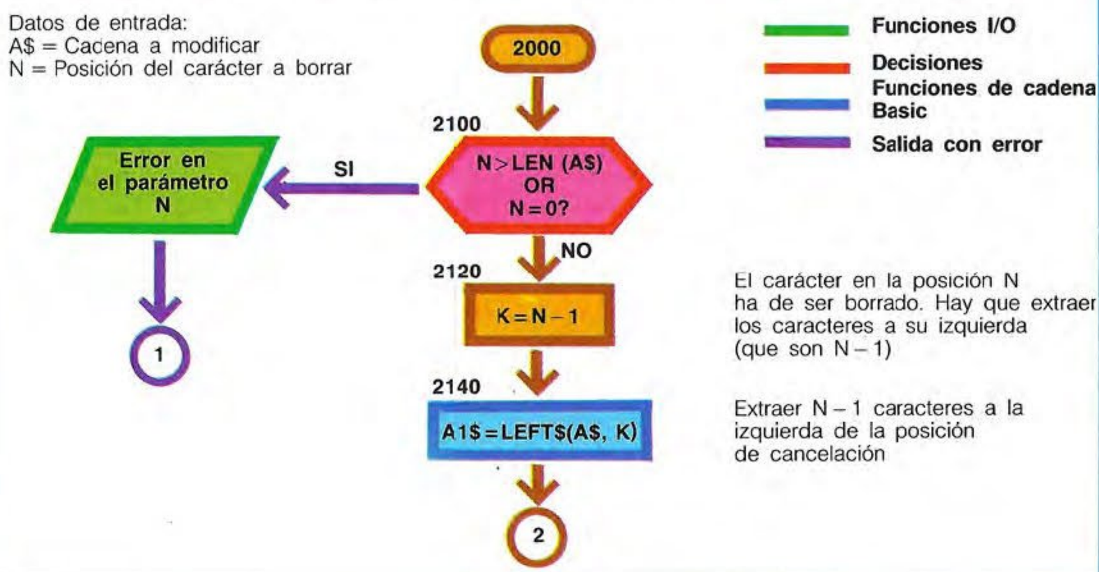


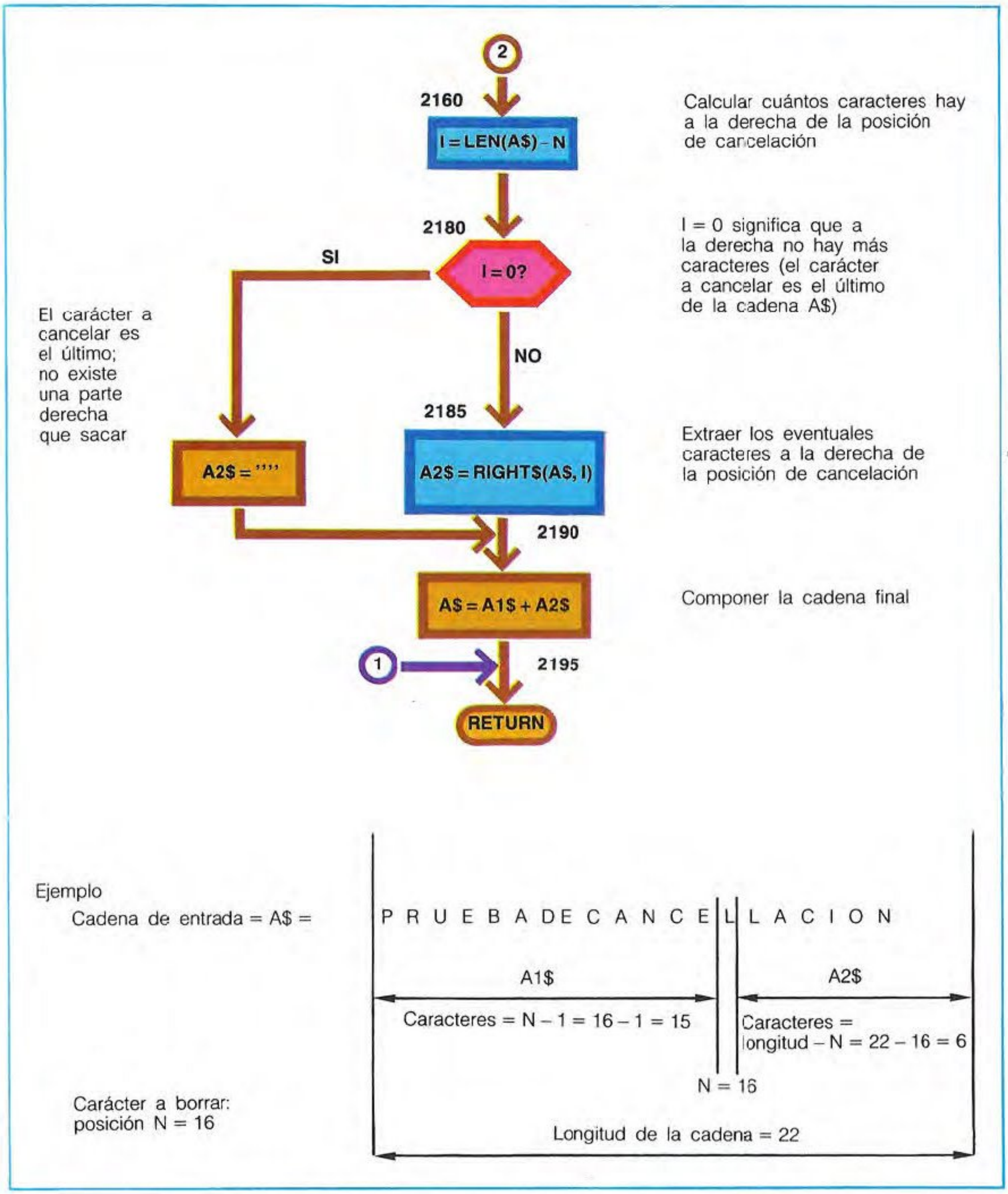
PROGRAMA DE APLICACION PARA EL USO DE LA SUBRUTINA 1000

```
50 INPUT "CADENA A MODIFICAR   ";A$
60 INPUT "CARACTER A INSERTAR  ";C$
62 IF LEN (C$)=0 THEN C$=" "
64 ' SI EL CARACTER A INSERTAR ES UN ESPACIO VACIO LA INSTRUCCION 62
66 ' SE HACE CARGO EN EL PROGRAMA
70 INPUT "INTRODUCIR EL NUM DE CARACT. QUE PRECEDEN AL QUE HAY QUE INSERTAR ";N
75 B$=A$      ' SALVO EN B$ LA CADENA ORIGINARIA (A MODIFICAR)
80 GOSUB 1000
90 '
100 '     INSTRUCCIONES DE IMPRESION
110 LPRINT "CADENA A MODIFICAR    ";B$
120 LPRINT "CARACTER A INSERTAR (";C$;")"
130 LPRINT "CADENA MODIFICADA : ";A$
132 LPRINT
134 GOTO 50
140 END
150 ' *** ATENCION: LAS SENTENCIAS 10-140 REPRESENTAN EL "MAIN" ***
1000 '** SUBRUTINA PARA LA INSERCION DE UN CARACTER EN UNA CADENA **
1001 '
1002 '
1003 '
1010 ' DATOS DE ENTRADA EN LA RUTINA :
1020 '                  A$ = CADENA A MODIFICAR
1030 '                  C$ = CARACTER A INSERTAR
1040 '                  N  = POSICION, EN NUMERO DE CARACTERES, A
1050 '                       EMPEZAR POR LA IZQUIERDA
1100 IF N>LEN(A$) OR N=0 THEN PRINT "ERROR EN EL PARAMETRO N" : GOTO 1400
1120 A1$=LEFT$(A$,N)    'MEMORIZAR EN (A1$) LOS CARACTERES A LA IZQUIERDA
1140 K=LEN(A$) ' CALCULAR LA LONGITUD DE (A$) Y MEMORIZARLA EN (K)
1145 I=K-N      ' I= NUMERO DE CARACTERES A LA DERECHA DE LA POSICION A INSERTAR
1160 IF I=0 THEN A2$="":GOTO 1220
1180 A2$=RIGHT$(A$,I)    'MEMORIZAR EN (A2$) LOS CARACTERES DE LA DERECHA
1220 A$=A1$+C$+A2$
1400 RETURN

CADENA A MODIFICAR : PRUEBA DE INSERCION
CARACTER A INSERTAR ( )
CADENA MODIFICADA: PRUEBA DE INSERCION
```

SUBRUTINA DE CANCELACION DE UN CARACTER EN UNA CADENA





STR$(R). Transforma el valor numérico R en una cadena en la que cada carácter representa una cifra del número. Así, el programa:

```
10 R = 126
20 A$ = STR$(R)
```

transfiere a la cadena A$ los caracteres 1, 2 y 6 representados en código ASCII.

Esta función puede usarse como alternativa a MKI$, MKS$ y MKD$ para la transferencia de valores numéricos al disco.

En las págs. 466, abajo, y 467, arriba, se muestra el diagrama de flujo de una subrutina que utiliza esta sentencia; en la pág. 467, abajo, se da el listado correspondiente.

En la subrutina se prevén algunas instrucciones que sirven para eliminar el primer carácter de la

PROGRAMA DE APLICACION PARA EL USO DE LA SUBRUTINA 2000

```
50 INPUT "CADENA A MODIFICAR   ";A$
60 INPUT "NUMERO DE POSICION DEL CARACTER A BORRAR   ";N
65 B$=A$         ' SALVO EN B$ LA CADENA ORIGINARIA (A MODIFICAR)
70 GOSUB 2000
80 '     INSTRUCCIONES DE IMPRESION
90 '
100 LPRINT "CADENA A MODIFICAR :  ";B$
110 LPRINT "CADENA MODIFICADA :  ";A$
120 LPRINT
130 GOTO 50
140 END
2000 ' ** SUBRUTINA DE CANCELACION DE UN CARACTER EN UNA CADENA **
2010 '     DATOS DE ENTRADA :
2020 '                        A$ = CADENA A MODIFICAR
2030 '                        N  = POSICION DEL CARACTER A BORRAR
2100 IF N>LEN(A$) OR N=0 THEN PRINT "ERROR EN EL PARAMETRO N":GOTO 2195
2120 K=N-1
2122 ' K= representa el número de caracteres a la izquierda
2124 '     del carácter a borrar
2140 A1$=LEFT$(A$,K)
2142 ' Memorizar en A1$ los caracteres a la izquierda del que hay que borrar
2160 I=LEN(A$)-N
2162 ' I= Calcular cuántos caracteres hay a la derecha de la posición de
2164 '     cancelación
2180 IF I=0 THEN A2$="":GOTO 2190
2185 A2$=RIGHT$(A$,I)
2190 A$=A1$+A2$
2195 RETURN

CADENA A MODIFICAR : PRUEBA DE CANCELACION
CADENA MODIFICADA : PRUEBA DE CANCELACION
```

SUBRUTINA DE CONVERSION DE NUMERICO A ASCII CON ELIMINACION DEL PRIMER CARACTER

Datos de entrada:
Z = Valor numérico a convertir en ASCII
N = Longitud de la cadena de salida

Datos de salida:
A$ = Cadena que contiene, en ASCII, el valor Z

Bucles y controles

Eventual llamada a la rutina de redondeo



En este punto se puede llamar la rutina de redondeo

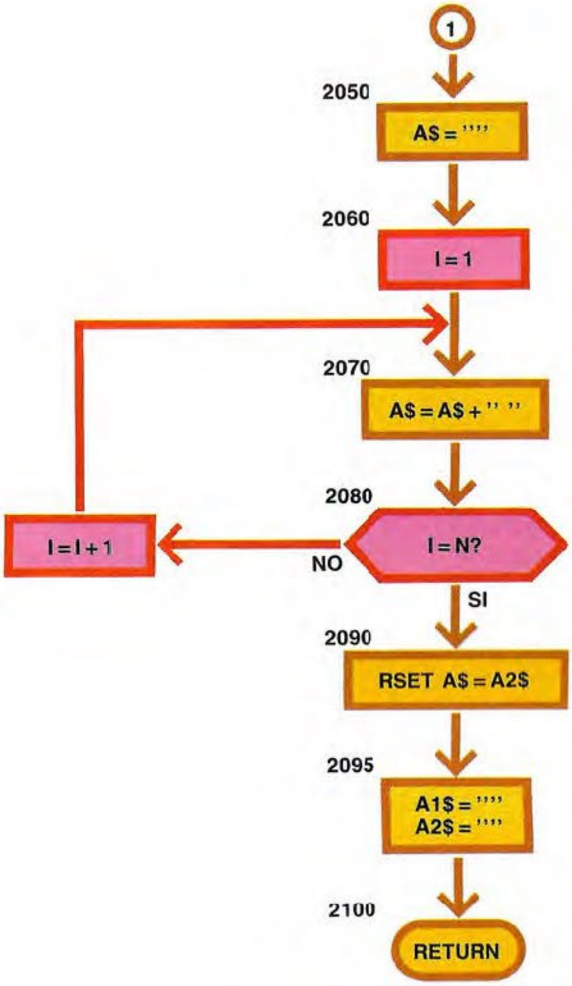
La cadena A$1 contiene el valor numérico Z convertido en ASCII

K (longitud de A1$) es igual al número de cifras más 1 (espacio en blanco)

A A2$ se transfieren K – 1 caracteres a partir de la derecha. De este modo A2$ contiene sólo los valores numéricos (el primer espacio vacío ha sido eliminado)

Controlar que el número de cifras no sea mayor que el máximo previsto

Si el número Z (y por tanto A2$) contiene más cifras que el máximo previsto, es truncado. El truncamiento se efectúa tomando a la izquierda, despreciando las cifras de la derecha, que son las menos significativas (contienen las unidades)



PROGRAMA DE APLICACION PARA EL USO DE LA SUBRUTINA 2000

```
50 INPUT "VALOR NUMERICO A CONVERTIR EN ASCII   ";Z
60 INPUT "LONGITUD DE LA CADENA DE SALIDA   ";N
70 Z1=Z          ' SALVO EN Z1 EL VALOR NUMERICO ORIGINARIO
80 GOSUB 2000
110 LPRINT "VALOR NUMERICO A CONVERTIR EN ASCII : ";Z1
120 LPRINT "VALOR TRANSFORMADO EN ASCII = ";A$
130 LPRINT
140 END
2000 ' ** SUBRUTINA DE CONVERSION DE NUMERICO EN (ASCII) **
2002 '     CON ELIMINACION DEL PRIMER CARACTER
2010 A1$=STR$(Z)
2020 K=LEN(A1$)
2030 A2$=RIGHT$(A1$, K-1)
2040 IF LEN(A2$)>N THEN A2$=LEFT$(A2$,N): GOTO 2050
2050 A$=""
2060 I=1
2070 A$=A$+" "
2080 IF I<>N THEN I=I+1: GOTO 2070
2090 RSET A$=A2$
2095 A1$=""
2096 A2$=""
2100 RETURN
```

TEST 14

1 / Algunas de las siguientes instrucciones son erróneas; ¿cuáles?
a) A$ = LEFT$(B$,4) b) A$ = MID$(B$,3)
c) A$ = STRING$(3,1) d) A$ = VAL("ABCHD")

2 / Una cadena contiene los siguientes caracteres (en total 38):
A$ = "Plumas tipo A cantidad 1250 color: rojo"
Sabiendo que la parte numérica (1250) comienza en la posición 24 y tiene 4 caracteres, escribir la subrutina para su extracción y conversión en valor numérico.

3 / Escribir una rutina que sume una cierta cantidad N al número obtenido como solución de la pregunta anterior y reconstruya la cadena con el valor actualizado.

4 / Qué sentencias o funciones hay que utilizar para obtener:
a) una cadena de longitud N formada toda ella por caracteres A
b) la representación octal y hexadecimal de un número N
c) la parte entera de un número real R

5 / Describir la salida del programa siguiente:

```
10 A$ = "*"
20 FOR I = 1 TO 20
30 B$ = SPACE$(I)
40 PRINT B$; A$
50 NEXT I
```

Las soluciones, en la pág. 473.

cadena A$. Efectivamente, el primer carácter es el signo, representado, para los valores numéricos positivos, por un espacio en blanco. En muchas aplicaciones no hay necesidad de conservar el signo (todos los valores son positivos) y conviene recuperar este espacio que, de lo contrario, se desperdiciaría.
Si el signo ha de conservarse, basta con eliminar las instrucciones de la 2010 a la 2030. En los programas de aplicación conviene adjudicar siempre la misma longitud a las cadenas que han de contener valores numéricos. La sentencia STR$, por el contrario, genera una cadena de longitud variable en función del número de cifras que ha de contener. Esta modalidad de funcionamiento no puede utilizarse en operaciones que impliquen a las unidades de disco, puesto que, como veremos, crea dificultades de encolumnamiento en la impresión y en la gestión de datos en pantalla.
El procedimiento a adoptar es el de establecer una longitud máxima de los números a tratar y dar dicha longitud a todas las cadenas.

Por ejemplo, estableciendo en ocho cifras la longitud máxima de los valores numéricos a convertir en cadena, cada conversión ha de generar una cadena de ocho caracteres, cualquiera que sea el número de cifras contenidas. En el ejemplo anterior, el valor numérico era 126, por lo tanto tendríamos una cadena de cuatro caracteres (el primero es un espacio en blanco). Para obtener un formato de ocho caracteres hay que preparar una cadena en blanco de dicha longitud, para luego transferir a ella la cadena que representa el valor numérico.
La creación de una cadena hecha de espacios en blanco puede lograrse simplemente con la sentencia SPACE$(N). En la rutina 2000, a título de ejemplo, la cadena se genera con un bucle.

STRING$(N,K). Crea una cadena de longitud N formada íntegramente por caracteres correspondientes al código (ASCII) K.
Por ejemplo, la línea:

A$ = STRING$(5,42)

genera la cadena A$ que contiene el símbolo * repetido cinco veces (el código ASCII 42 corresponde a dicho símbolo).
La sentencia existe también en la forma:

STRING$(N,B$)

En este caso se genera una cadena de N caracteres iguales al primer carácter de B$. Por ejemplo, las líneas:

```
10 B$ = "***ABC***"
20 A$ = STRING$(5,B$)
```

generan una cadena (A$) igual a la del ejemplo anterior, puesto que el primer carácter de B$ es el mismo indicado por el código 42 en el ejemplo anterior.

VAL(A$). Da el valor numérico de la cadena A$. En otras palabras, es la función inversa de STR$(R).
Por ejemplo:

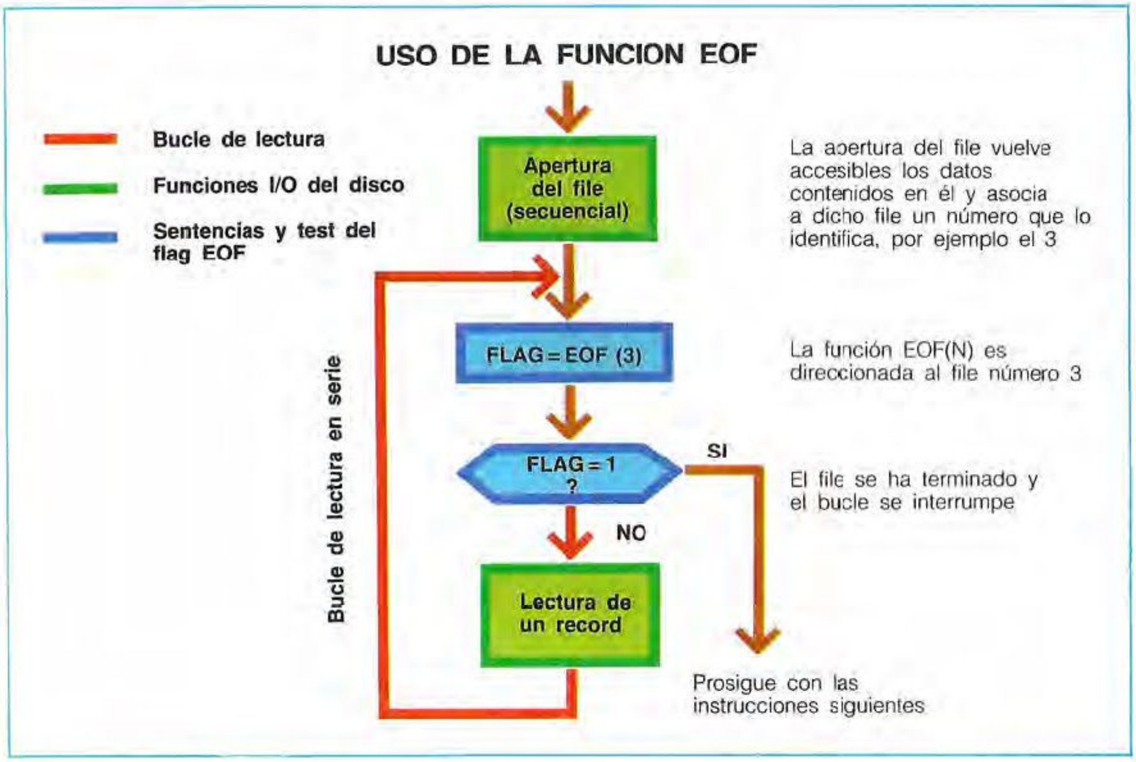
```
10 A$ = "576.51"
20 R = VAL(A$)
```

El contenido de A$ es transformado en valor numérico y transferido a la variable R.
Esta función también se usa principalmente para la transferencia de los datos del disco.

Funciones especiales

A este grupo pertenecen algunas funciones dedicadas a la gestión de los periféricos o de la memoria. Su uso correcto implica el conocimiento de temas que expondremos más adelante, por lo que las comentamos en este capítulo para completar la enumeración.

EOF(N). Proporciona un flag con valor –1 cuando, durante la lectura de un file secuencial, se llega a su fin.
La lectura de un file secuencial puede efectuarse con un bucle que toma sucesivamente todos los records. Para señalar el final del file y, por tanto, para interrumpir el bucle, hay que utilizar la sentencia EOF, de lo contrario habría un intento de lectura más allá de la longitud máxima del file, con la consiguiente señalización de error e interrupción del programa.
Abajo se muestra el diagrama de flujo que ilustra, desde el punto de vista lógico, el uso de esta función. En el capítulo dedicado a la gestión de los files se dan ejemplos de aplicación.
El valor N incluido en la función indica con un número el file al que se refiere la EOF. Por ejemplo, si ponemos EOF(3) la función se direcciona



al file número 3 (el número se asocia al file en la fase de «apertura»).

FRE(A). Da la amplitud, en número de bytes, de la memoria aún disponible. El argumento A es ficticio, es decir, no tiene significado alguno, y se pone únicamente para respetar la sintaxis de la función.

Para ejemplificar el uso de la función FRE, supongamos que tenemos una máquina de 36.000 bytes de memoria, y que hemos cargado un programa que ocupa 14.000. La función FRE(A) da el valor 22.000, que es el área de memoria aún disponible.

Como se ha dicho, el argumento A es ficticio, y puede sustituirse por un símbolo cualquiera; así, las sentencias FRE(A), FRE(O) y FRE(C$) dan el mismo resultado.

La sintaxis completa de la sentencia que hay que introducir para obtener escrita en pantalla la amplitud del área de memoria disponible es PRINT FRE(A).

INP(N). Es una función de entrada genérica. Proporciona el valor del byte presente en la «puerta» número N. Así, por ejemplo:

VAL = INP(2)

toma un byte de la puerta 2 y lo transfiere a la variable VAL. El valor obtenido es un número entero comprendido entre 0 y 255 (con un byte se puede escribir, como máximo, el número decimal 255).

Esta función (como también su «complementaria» OUT) se usa sólo para aplicaciones especiales; la sentencia normal para introducir datos mediante el teclado es INPUT.

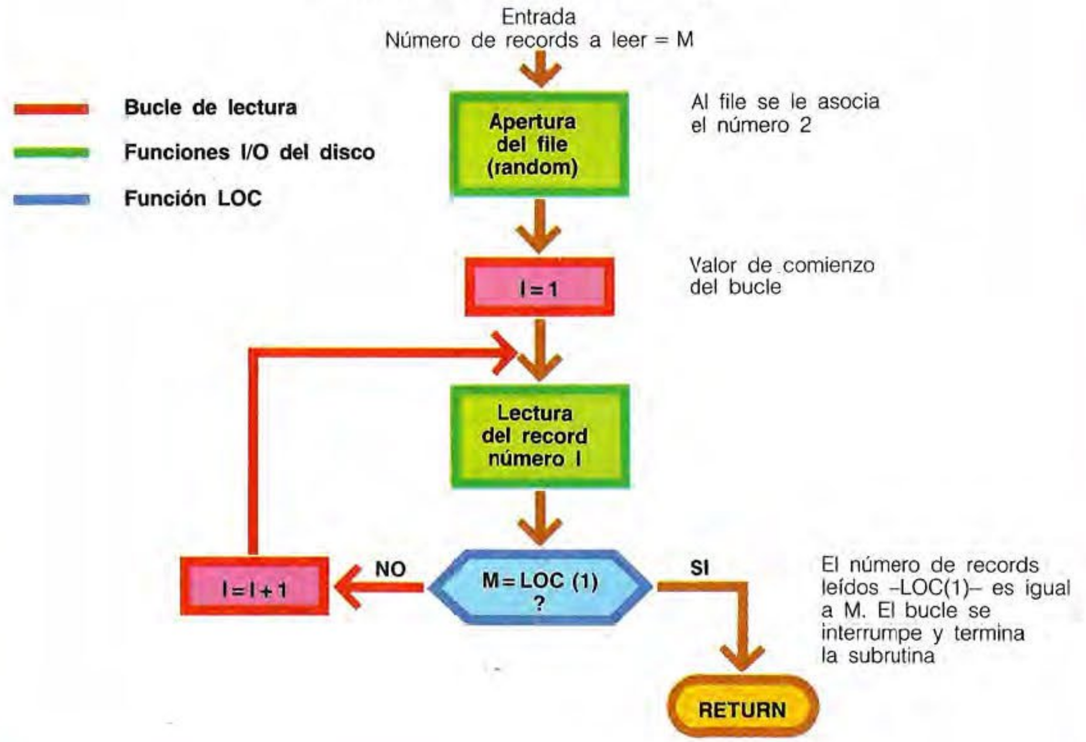
LOC(N). Restituye el número del último registro leído o escrito en un file random.

Abajo se muestra el diagrama de flujo de una subrutina para la lectura de un file random de un número prefijado M de records.

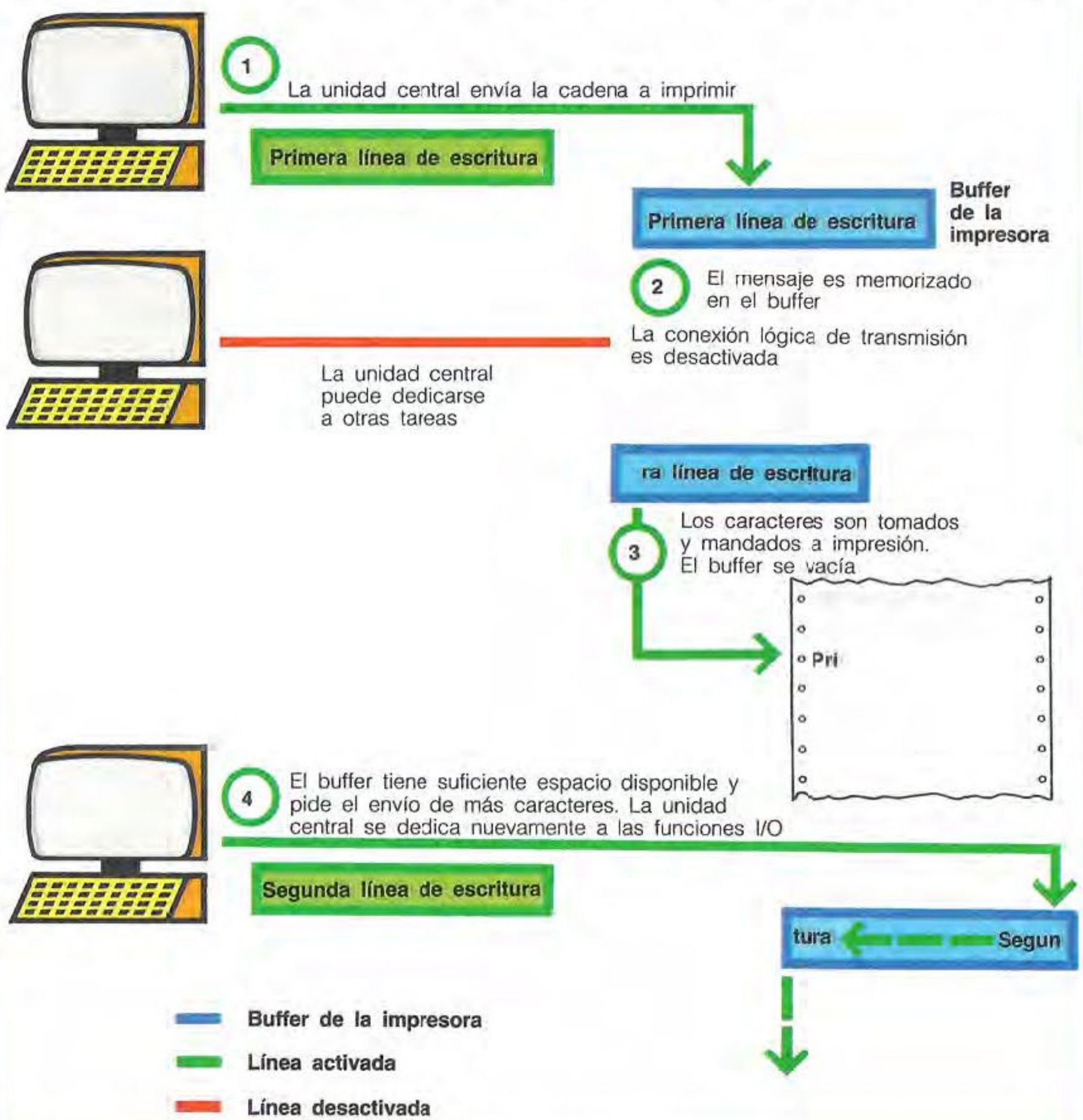
El test de final del bucle se efectúa con la función LOC, pero podría realizarse con el índice del bucle con los mismos resultados.

El ejemplo no es una aplicación real, sino que constituye tan sólo la ilustración del uso de la función LOC.

USO DE LA FUNCION LOC



TRANSFERENCIA DE DATOS DE LA UNIDAD CENTRAL A LA IMPRESORA



LPOS(N). Restituye la posición que ocupa en el buffer de la impresora el carácter en curso de impresión.

Las impresoras, como otros periféricos, disponen de un buffer, es decir, una memoria local que se carga con los caracteres a imprimir. La unidad central envía un determinado número de caracteres, que se acumulan en el buffer; terminada la transferencia, la unidad central puede dedicarse a otras tareas, mientras la impresora toma los caracteres del buffer y los imprime, con un funcionamiento totalmente indepenciente del de la unidad central.

A medida que se imprimen los caracteres, el buffer se vacía. Cuando en el buffer queda espacio libre suficiente, la impresora envía un mensaje a la unidad central pidiendo más datos. De esta forma se reducen los tiempos muertos debidos a las esperas para la cumplimentación de las operaciones de impresión.

En el gráfico superior se esquematiza la lógica descrita. Téngase en cuenta que LPOS(N) da la posición en el buffer, que puede no coincidir con la posición física de la cabeza de escritura a causa de las enormes diferencias de velocidad entre la transferencia de un carácter (que

viene representado por señales eléctricas) y su impresión sobre papel, fase que implica el movimiento de órganos mecánicos.
Precisamente a causa de la presencia de órganos mecánicos, la impresora es el componente más lento de todo el sistema, e incluso con los tipos más sofisticados (impresoras a rayo láser) las velocidades que se pueden obtener no son siquiera comparables a las de elaboración*.

OUT N,M. Es una instrucción, pero se incluye en el capítulo de las funciones ya que realiza una tarea «recíproca» con respecto a la INP(M).
Esta sentencia permite transferir a la puerta de salida N el valor numérico M. Por ejemplo:

```
OUT 3,127
```

transfiere el valor numérico 127 a la puerta de salida número 3. Esta sentencia también se usa sólo para tareas especiales, puesto que el Basic prevé para las funciones de emisión de datos normales otros tipos de instrucciones.
La sentencia OUT y la función INP están muy próximas a la forma utilizada en el lenguaje Assembler. La sentencia Basic INPUT (que es una sentencia de entrada de datos como INP) hace superfluo especificar a qué periférico se pide la introducción de datos, ya que el intérprete Basic y el compilador asignan automáticamente esta operación al teclado. Por el contrario, en Assembler (y parcialmente también en Fortran) la asignación no es automática: ha de ser el programador quien especifique el número de la puerta (o de la unidad de entrada) por la que se desea la introducción. Esto contribuye a explicar cuáles pueden ser los motivos que llevan, en algunas aplicaciones, a usar el lenguaje Assembler en lugar del Basic.
Los microordenadores modernos disponen de diversas puertas de entrada, con interfaces adecuados para realizar funciones especiales, por ejemplo, capaces de comunicar con otros ordenadores mediante una línea telefónica. Programando en Basic estándar no se pueden utilizar estos dispositivos especiales.
Ni siquiera las sentencias INP y OUT sirven para estas aplicaciones especiales (como la transmisión de datos) que suponen una notable movilización de la estructura del hardware de la máquina. El lenguaje más adecuado, puesto que es el más próximo a la estructura de la máquina, es en estos casos el Assembler.
Sólo en algunas máquinas muy evolucionadas, pertenecientes a la gama más perfeccionada de los microordenadores, hay sentencias utilizables en Basic que gestionen de forma completa los interfaces. Se trata, en cualquier caso, de máquinas destinadas a usos específicos, como el control automático de aparatos, la toma de datos a partir de instrumentos, etc.

PEEK(N). Restituye el contenido de la memoria número N. El valor obtenido es un número comprendido entre 0 y 255 (una memoria corresponde a un byte). La función PEEK sólo puede utilizarse si se conoce la dirección «absoluta» del dato buscado.
En Basic (y en todos los lenguajes simbólicos) cada memoria de datos tiene un nombre, mediante el cual se utiliza en los programas. La dirección a la que corresponde dicha memoria (es decir, su posición en el área de memoria) no es conocida para el programador; por lo tanto, la función PEEK (así como su sentencia recíproca POKE) se utiliza escasamente en los programas de aplicación.
Una utilización típica de esta función (y de la POKE) estriba en la gestión de la pantalla.
Una parte de la memoria de la máquina se dedica a la pantalla. En esta zona de memoria se conservan los «atributos» de la pantalla, es decir, algunas magnitudes características, como, por ejemplo, la forma del cursor.
Conociendo el «mapa de memoria», es decir, la posición en que están contenidos estos datos, se pueden efectuar modificaciones, como cambiar la forma del cursor y la frecuencia con que parpadea.
La función PEEK (y POKE) no representa, sin embargo, la única forma de acceso a la gestión del vídeo: hay funciones utilizables en Basic que realizan la misma tarea de manera más completa, y que se expondrán más adelante.

POKE N,M. También la sentencia POKE, que no representa una función sino una instrucción, se incluye aquí puesto que opera de forma recíproca a la de la función PEEK. La sentencia POKE N,M escribe en la memoria número N el valor M, con M comprendido entre 0 y 255.

* La velocidad de elaboración se mide en MIPS (Millones de Instrucciones Por Segundo); las máquinas modernas pueden llegar a 1-2 MIPS, y ya están en fase de estudio y experimentación tecnologías que permitirán la fabricación de sistemas mucho más veloces.

SOLUCIONES DEL TEST 14

1 / Las instrucciones erróneas son la b) y la d). La función MID$ necesita dos valores enteros, por ejemplo en la forma MID$(A$,3,2). El primer entero es la posición inicial de la que parte la extracción, y el segundo es el número de caracteres a tomar.
En la d), la función VAL se aplica a una cadena de caracteres, cuando sólo opera con cadenas de números. En cualquier caso, el valor dado por la función, que es numérico, no puede asignarse a la cadena A$.

2 / Se pueden usar la función MID$(A$,24,4) para extraer la parte numérica y la función VAL para convertirla en número. La subrutina se reduce, pues, a dos instrucciones:

```
1000 '* Entrada en la subrutina
1010 B$ = MID$(A$,24,4)
1020 M = VAL(B$)
1030 RETURN
```

3 / Esta segunda subrutina es más compleja que la anterior, puesto que para sustituir el resultado de la suma (N + M, donde N = nuevo valor y M = valor extraído en la subrutina anterior) hay que dividir A$ en tres partes: la de la izquierda del número, el número convertido en cadena y la parte de la derecha. Volviendo a unir las tres partes se obtiene nuevamente la cadena A$ completa y con el valor numérico sustituido.

```
2000 '* *                    Entrada en la subrutina
2010 AS$ = LEFT$(A$,23)      'Parte izquierda
2020 AN$ = STR$(N + M)       'Valor numérico
2030 AD$ = MID$(A$,28,14)    'Parte derecha
2040 A$ = AS$ + AN$ + AD$
2050 RETURN
```

Observando que los caracteres a la derecha del número son 41 – 27 = 14, la instrucción 2030 puede sustituirse por AD$ = RIGHT$(A$,14).

4 / a) STRING$(N,A$), habiendo puesto A$ = "A"
b) OCT$(N) y HEX$(N)
c) FIX(R); atención a la diferencia con INT(R)

5 / La función SPACE$(I) crea una cadena de caracteres en blanco (space). En fase de impresión se tiene, para cada línea, un asterisco desplazado una posición hacia la derecha respecto a la línea anterior. A la primera vuelta en el bucle (I = 1) se imprime primero una cadena que contiene un solo espacio [B$ = SPACE$(1)], luego A$, que por tanto ocupa la columna 2. A la segunda vuelta, B$ contiene dos espacios y A$ se imprime en la columna 3, y así sucesivamente.

POS(N). Restituye la posición del cursor. El parámetro N es ficticio. Por ejemplo, la línea:

K = POS(0)

transfiere a la variable K la posición actual del cursor.

Funciones definidas por el usuario

El repertorio estándar de las funciones intrínsecas al lenguaje Basic puede enriquecerse añadiendo, en el seno de los programas aplicativos, nuevas funciones que el usuario puede definir. La sentencia a utilizar para definir una nueva

función ha sido ya examinada ocasionalmente, y la repetimos ahora:

DEF FN NOMBRE = expresión

NOMBRE indica el nombre de una variable genérica, y «expresión» representa el cálculo que la función ha de efectuar para obtener la variable de salida.

Por ejemplo, si se desea calcular la hipotenusa H de un triángulo rectángulo cuyos catetos son X e Y, hay que realizar el siguiente cálculo (teorema de Pitágoras):

$$H = \sqrt{X^2 + Y^2}$$

que, traducido al Basic (la raíz se calcula con la función SQR, y la potencia se indica con el símbolo Λ), se convierte en:

H = SQR(XΛ2 + YΛ2)

Si fuera necesario efectuar este cálculo varias veces a lo largo de un programa, tendríamos tres posibilidades:

1 / reescribir la fórmula cada vez
2 / utilizar una subrutina
3 / utilizar una función

La primera solución no es racional, puesto que aumenta la ocupación de memoria; utilizar una subrutina que contiene una sola línea (el cálculo de H) es incómodo, pues para poder llamarla hay que recordar el número de la línea en que ha sido escrita (la llamada a una subrutina es GOSUB nnn, donde nnn es el número de la línea en que empieza la subrutina). La última solución es la mejor. Escribiendo:

DEF FNA(X,Y) = SQR(XΛ2 + YΛ2)

se crea una función de dos variables, X e Y (catetos), de nombre A, que efectúa el cálculo de la hipotenusa. Poniendo en cualquier punto del programa:

H = FNA(X,Y)

se obtiene en la variable H el valor de la hipotenusa. En la pág. 475 se muestran los diagramas

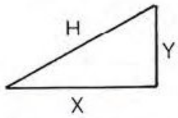
Centro de proceso de datos que utiliza equipos IBM.



K. Reese/Marka

REALIZACION DE UN CALCULO CON EL USO DE UNA SUBRUTINA Y DE UNA FUNCION

El cálculo a realizar es
$H = \sqrt{X^2 + Y^2}$



Instrucciones inherentes a la función
Funciones I/O
Subrutina

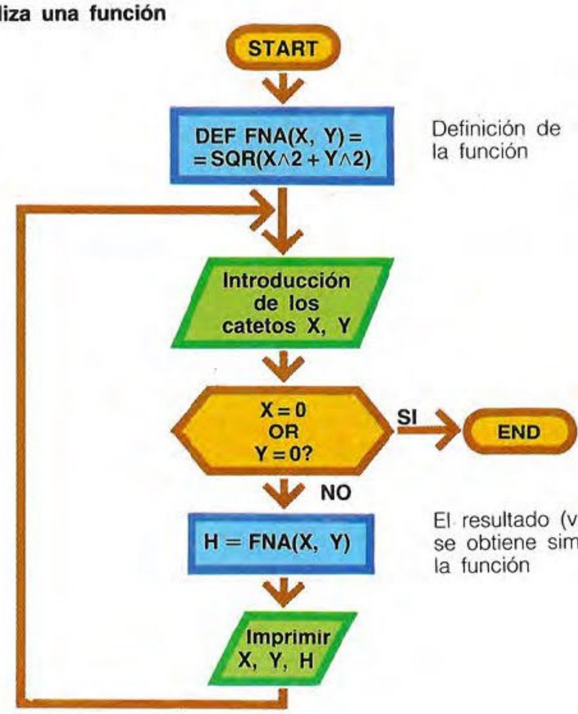
Programa que utiliza una subrutina



El programa termina poniendo uno de los catetos igual a cero

La subrutina realiza el cálculo H = SQR(X∧2 + Y∧2)

Programa que utiliza una función



Definición de la función

El resultado (valor de la hipotenusa) se obtiene simplemente llamando la función

de flujo de dos programas que efectúan este mismo cálculo. El primero utiliza una subrutina; el segundo, una función. Abajo se ven los listados correspondientes y algunas salidas.
La función que acabamos de definir es de tipo numérico (los parámetros y el resultado són números), pero de igual forma se pueden definir funciones de cadena. Éstas tienen cadenas de caracteres como parámetros y dan una cadena como resultado. Así, la función:

DEF FNA$(C$,N) = LEFT$(C$,N) + D$

PROGRAMA QUE UTILIZA UNA SUBRUTINA

```
10 ' **PROGRAMA QUE UTILIZA UNA SUBRUTINA **
20 '                FILE = PROSUB
30 PRINT "INTRODUCIR LOS DOS CATETOS (X-Y)"
40 INPUT "X = ";X
50 INPUT "Y = ";Y
60 IF X=0 OR Y=0 GOTO 140
70 GOSUB 200    ' EN LA 200 ESTA LA SUBRUTINA QUE CALCULA LA HIPOTENUSA
80 ' INSTRUCCIONES DE IMPRESION
90 LPRINT "PRIMER CATETO:          X =";X
100 LPRINT "SEGUNDO CATETO:        Y =";Y
110 LPRINT "HIPOTENUSA:            H =";H
120 LPRINT
130 GOTO 30              ' RETORNO A OTRO CALCULO
140 END
150 '
160 ' EL ( MAIN ) VA DE LA LINEA 10 A LA LINEA 140
170 '
180 '
190 '
200 ' * SUBRUTINA PARA EL CALCULO DE LA HIPOTENUSA (CONOCIENDO LOS CATETOS)
210 '
220 H=SQR(X*X+Y*Y)  ' Transferir a la variable H el cálculo de la raíz
230                   cuadrada (SQR) de la suma de los cuadrados de los catetos
240 '
250 RETURN
```

```
PRIMER CATETO       X = 2
SEGUNDO CATETO      Y = 2
HIPOTENUSA          H = 2.82843
```

PROGRAMA QUE UTILIZA UNA FUNCION DEFINIDA POR EL USUARIO

```
10 ' **PROGRAMA QUE UTILIZA UNA FUNCION DEFINIDA POR EL USUARIO **
20 '                       FILE = PROFUN
30 DEF FNA(X,Y)=SQR(X*X+Y*Y)   ' El usuario ha definido una función que se
40 '                             denomina A(X,Y)
50 PRINT "INTRODUCIR LOS DOS CATETOS (X-Y)"
60 INPUT "X = ";X
70 INPUT "Y = ";Y
80 IF X=0 OR Y=0 GOTO 160
90 H=FNA(X,Y)       ' TRANSFERIR A (H) EL RESULTADO DE LA FUNCION FNA(X,Y)
100 ' * INSTRUCCIONES DE IMPRESION *
110 LPRINT "PRIMER CATETO:     X =";X
120 LPRINT "SEGUNDO CATETO:    Y =";Y
130 LPRINT "HIPOTENUSA:        H =";H
140 LPRINT
150 GOTO 50              RETORNO A OTRO CALCULO
160 END
```

```
PRIMER CATETO:       X = 2
SEGUNDO CATETO:      Y = 2
HIPOTENUSA:          H = 2.82843

PRIMER CATETO:       X = 4
SEGUNDO CATETO:      Y = 3
HIPOTENUSA:          H = 5
```

Un ordenador tras las ruedas

Con el descubrimiento de los microcircuitos integrados y con su constante perfeccionamiento, ya no hay ningún obstáculo técnico que se oponga a la instalación en un automóvil de un ordenador centralizado, cuyas dimensiones no se opongan a las exigencias de funcionalidad y cuya potencia, al mismo tiempo, permita la total automación de las funciones del vehículo. El verdadero problema reside en la «interface» entre el automóvil y su ordenador.

Para que el ordenador pueda realizar sus funciones, ha de estar siempre meticulosa y rápidamente informado de todos los factores que determinan su decisión final. Sin embargo, el gran número de funciones que un automóvil ha de realizar y la cantidad aún mayor de parámetros que las determinan, hacen muy complejo el funcionamiento del ordenador. El microprocesador de un sistema de inyección del combustible, por ejemplo, exige datos sobre la velocidad del motor, la carga, la posición de las válvulas, la temperatura y la densidad del aire para poder suministrar la mezcla correcta en todas las circunstancias. Cada uno de estos valores, además, requiere un sensor propio conectado con el microprocesador; no es necesario que estos sensores sean especialmente complejos, pero es evidente que su número aumenta al aumentar las funciones bajo control.

Además, todos estos sensores han de estar adecuadamente conectados al ordenador; en otras palabras, tanto si el sensor registra impulsos en forma de variaciones de resistencia eléctrica, como si los registra en forma de señales procedentes de una fotocélula o de cualquier otra cosa, éstos han de ser traducidos a la forma digital. La información procedente de los sensores casi nunca está en forma digital en origen; normalmente llega en forma «analógica», es decir, la señal suministrada en la salida cambia al variar la magnitud que se está midiendo, dando así una señal que varía continuamente, y no una secuencia de impulsos separados. Sin embargo, el paso de la forma analógica a la digital no es muy difícil.

Los problemas relativos a las conexiones de entrada con el interface del ordenador son, de hecho, mucho menos arduos que los relativos a la interface externa, es decir, la que convierte las decisiones del ordenador en maniobras del automóvil. Este es el punto en el que una de las ventajas de los microcircuitos integrados, su bajísimo consumo energético, se transforma en una gran desventaja. El inconveniente consiste en el hecho de que una señal de un valor de una minúscula fracción de amperio ha de accionar una voluminosa válvula de alimentación del combustible o actuar sobre los frenos con la fuerza suficiente para detener el automóvil; obviamente, en determinados casos, la señal ha de ser notablemente amplificada. Como en los amplificadores de alta fidelidad, la amplificación se obtiene utilizando una señal débil para disponer de una más fuerte; en un sistema de circuito cerrado, esto puede crear oscilaciones indeseadas, que pueden surgir cuando una corrección empiece a modificar un error; en el peor de los casos, la oscilación puede desviarse y rectificar en exceso, cosa que, mientras que en un sistema de Hi-Fi produciría un ruido estridente, en el sistema de frenado de un automóvil podría traducirse en un accidente mortal. Incluso cuando la generación es muy limpia y sin oscilaciones incontroladas, la señal puede no resultar correcta. Los frenos de los automóviles actuales, por ejemplo, pueden funcionar a presión hidráulica; por lo tanto, la interface para la parte del ordenador destinada al control del sistema de frenado, ha de incluir no sólo un amplificador, sino también un dispositivo para convertir un impulso eléctrico en un movimiento hidráulico.

Es evidente que los problemas de interface representan un enorme obstáculo para la computerización de los automóviles, pues son mayores que los del ordenador en sí; efectivamente, mientras que los sistemas de elaboración han llegado ya a un buen nivel y sus capacidades son bien conocidas, los problemas de las interfaces son los que requieren mayor atención en las investigaciones sobre sistemas electrónicos para los automóviles.

Por todo ello, la utilización de ordenadores en los automóviles sólo se ha dado hasta ahora a escala reducida. Las empresas automovilísticas han instalado microprocesadores en los modelos más avanzados de su producción a fin de darles un mayor rendimiento y hacerlos más competitivos o como reclamo publicitario. Por ello la introducción de microprocesadores ha ido ligada a tres funciones principales: carburación, encendido e instrumentación.

Hoy día existen ya muchos tipos de automóvil con encendido electrónico y, si bien la inmensa mayoría tiene todavía anticipación mecánica, un

cierto número de sistemas más recientes (para mejorar la precisión de la anticipación en el encendido de las bujías) han empezado a introducir microprocesadores a los que se transmiten datos relativos a la velocidad del motor, la posición del pistón y la carga sobre el motor, datos todos ellos que son confrontados con las instrucciones introducidas en la memoria del microprocesador en la fase de fabricación. La comparación permite al microprocesador decidir el momento del encendido y dar las instrucciones necesarias para que éste tenga lugar en el instante preciso. Ya se ha comprobado que este tipo de dispositivo electrónico mejora las prestaciones del automóvil y reduce la emisión de gases de escape; concretamente, Volkswagen ha demostrado que es posible mantener el motor en un régimen de mínimo estable variando la puesta a punto.

Estos sistemas, sin embargo, constituyen solamente el punto de partida, puesto que las posibilidades de control mediante ordenador son muy superiores. A partir de finales de los setenta, por ejemplo, algunas empresas automovilísticas han introducido un «sensor de detonación» cuya importancia consiste en el hecho de que los motores podrían ser más eficientes si giraran con una relación de compresión más elevada. Actualmente, los motores a gasolina difícilmente superan la relación de compresión de 10:1, por otra parte disminuida a menudo por los límites impuestos al porcentaje de plomo en la gasolina; además, si dicha relación fuera mayor, el motor estaría sujeto a detonación (knocking) *por lo que, cuando ésta fuera a producirse, una parte de la mezcla, antes de ser inflamada, explotaría con violencia, en vez de arder simplemente, con los consiguientes daños irreparables para el motor y pérdida de potencia. El sensor destinado a la detonación la advierte poco antes de que tenga lugar gracias a ciertos signos «premonitorios», tales como una insólita variación en la velocidad del eje, retrasando el encendido para prevenir el efecto detonante.*

Parecería que una vez dada al motor la posibilidad de funcionar, gracias al empleo de microprocesadores, con la máxima economía de combustible en la mezcla, con una relación de compresión muy elevada y con emisiones de gas casi nada contaminantes, a la electrónica le quedaría bien poco por hacer para mejorar las prestaciones del motor. Sin embargo, no todos son de esta opinión, ya que prevén en el microprocesador mayores capacidades potenciales: las válvulas, por ejemplo, constituyen una gran posibilidad para la tecnología electrónica, tanto es así que Lucas-CAV ha realizado ya, en Gran Bretaña, válvulas accionadas electromagnéticamente, y desde 1980 existe en Estados Unidos un prototipo de motor dotado de válvulas hidráulicas.

El funcionamiento electrónico de las válvulas abre un enorme campo de posibilidades: podría, por ejemplo, permitir al conductor modificar las características del motor a voluntad, modulando la conducción mediante un conmutador, con una simple variación de la puesta a punto y de la duración de la apertura de las válvulas; por otra parte, el conductor podría provocar el cierre simultáneo de todas las válvulas, transformando el motor en un compresor, para una más eficaz acción de frenado. Un sistema tal vendría, con toda probabilidad, a ampliar las prestaciones de un microordenador más que cualquiera de las aplicaciones actuales, que aprovechan sólo parcialmente sus innumerables capacidades. El motor no es en absoluto la única parte del automóvil que puede beneficiarse de la electrónica. Otro sector prometedor es el de la transmisión, y son muchos los ingenieros convencidos de que en un futuro próximo se obtendrá mayor economía en el consumo por el control electrónico de la transmisión que por la aplicación de microprocesadores al motor. En un sistema de este tipo, en el microprocesador podrían memorizarse las características del motor y ello modificaría automáticamente las relaciones, para adaptarlas a la velocidad y a las condiciones de carga; sin embargo, es probable que para una confirmación definitiva haya que esperar a ver los adelantos en el campo de la transmisión continua (CVT); en uno u otro caso, el control mediante microprocesadores redundará en una adaptación más precisa de la relación de transmisión con respecto a la potencia y a la velocidad. Sin embargo, la señal más evidente de la presencia de la electrónica en los automóviles de hoy está, en muchos aspectos, en el cuadro de instrumentos; tras la aparición del Lagonda (que tuvo lugar en 1976 por obra de la reestructurada Aston Martin Company), provisto de instrumentación en estado sólido y de un panel de visualización totalmente digital, han aparecido en el mercado un cierto número de automóviles equipados de forma similar, con paneles parecidos a los de una astronave. Estos

visualizadores no ofrecen ninguna ventaja intrínseca (los cuadrantes tradicionales indican igual, si no mejor, la velocidad, el número de revoluciones del motor, el nivel de combustible, etc.), pero abren el camino a la introducción de una amplia gama de funciones; efectivamente, algunos automóviles disponen ya de un indicador que permite un constante control visual del consumo de combustible y de la velocidad media. Los diseñadores sugieren la posibilidad de proyectar una indicación digital sobre el parabrisas de forma que el conductor disponga de todos los datos que necesita sin tener que apartar la vista de la calzada; indicadores frontales (HUD) de este tipo estuvieron en uso durante algunos años en los cazas a reacción, pero todavía no se ha logrado construirlos lo suficientemente pequeños y de costo moderado como para poder instalarlos en los automóviles; tampoco se ha encontrado una solución capaz de suministrar la iluminación necesaria para que estos visualizadores resulten fácilmente legibles incluso en días soleados.

Otro campo al que no se ha prestado la debida atención es el sistema de los frenos y el de la suspensión. Actualmente, sin embargo, BMW y Mercedes ofrecen en sus modelos más avanzados un sistema de frenado antibloqueo. Considerando que hace más de veinte años que dichos sistemas se usan en los aviones, es sorprendente que su introducción en el campo automovilístico haya llevado tanto tiempo; el problema era que, mientras que un sensor hidromecánico y un sistema de control actúan con bastante eficacia sobre las ruedas de un pesado avión, no responden con la suficiente rapidez al sistema de frenado mucho más ligero del automóvil medio. En un sistema electrónico antibloqueo, el sensor advierte que la rueda está disminuyendo de velocidad demasiado deprisa (es decir, que va a bloquearse) y el dispositivo se encarga de que los frenos se aflojen rápidamente para evitar la detención de la rueda y que, con igual prontitud, vuelvan a funcionar.

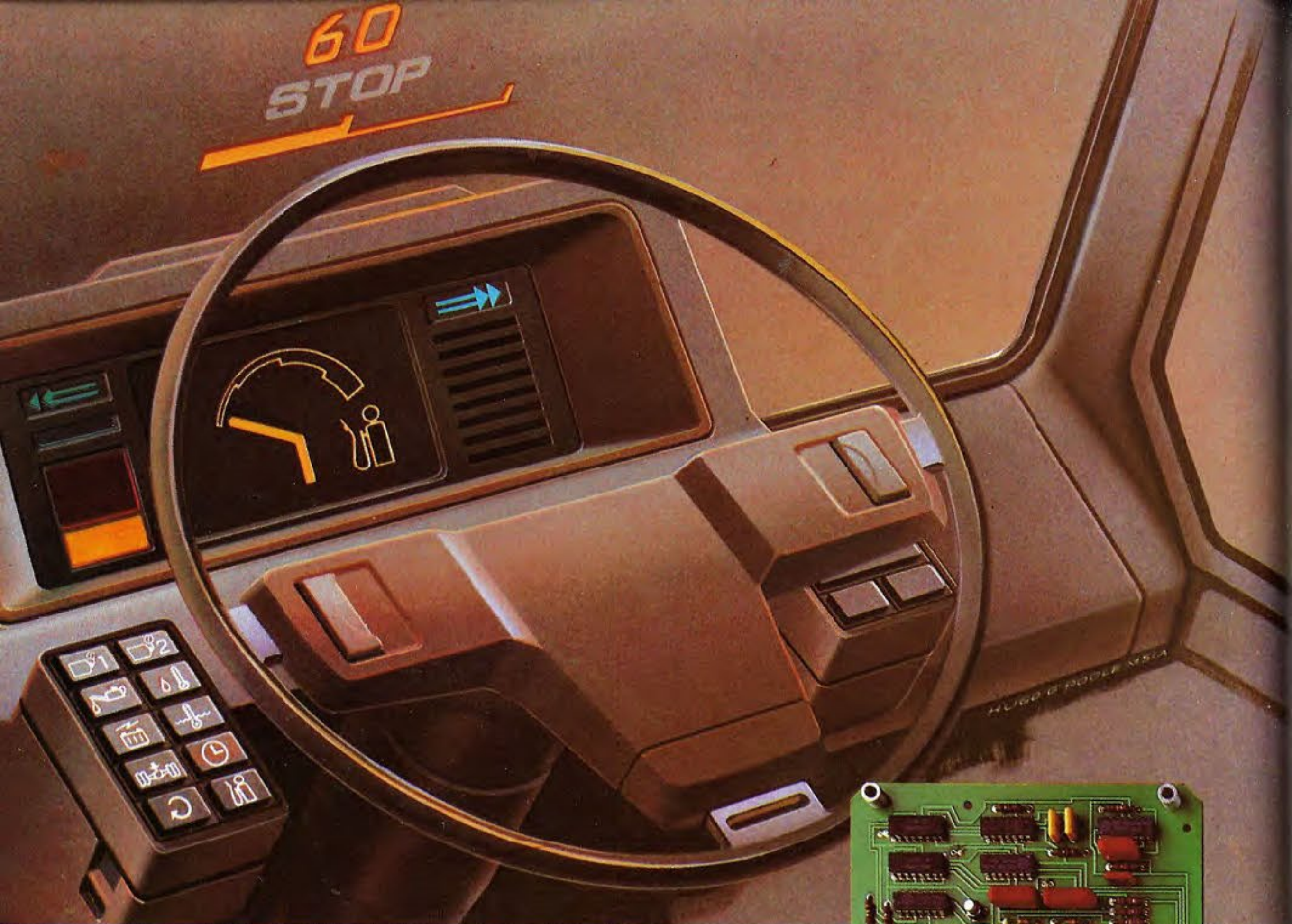
Algunos ingenieros opinan que si los frenos pueden controlarse de esta forma mediante conductos hidráulicos, una operación similar podría efectuarse también con la suspensión. A este respecto ya se han probado sistemas que impiden al vehículo oscilar en una curva, pero una solución totalmente electrónica podría conseguir mucho más. La mayor parte de los sistemas de suspensión tradicionales están limitados por el tipo de muelles con que se construyen: un muelle ideal tendría que actuar gradualmente, es decir, ceder fácilmente al comienzo, de forma que absorbiera las pequeñas sacudidas, y volverse poco a poco más rígido para hacer frente a las fuertes presiones a que está sometido. Esto es difícil de obtener con muelles mecánicos, pero es más fácil con los de aire-líquido, los cuales, junto con un microprocesador que controle una válvula al objeto de dejar salir el aire del muelle o bombearlo en su interior, suministrarían una suspensión excelente.

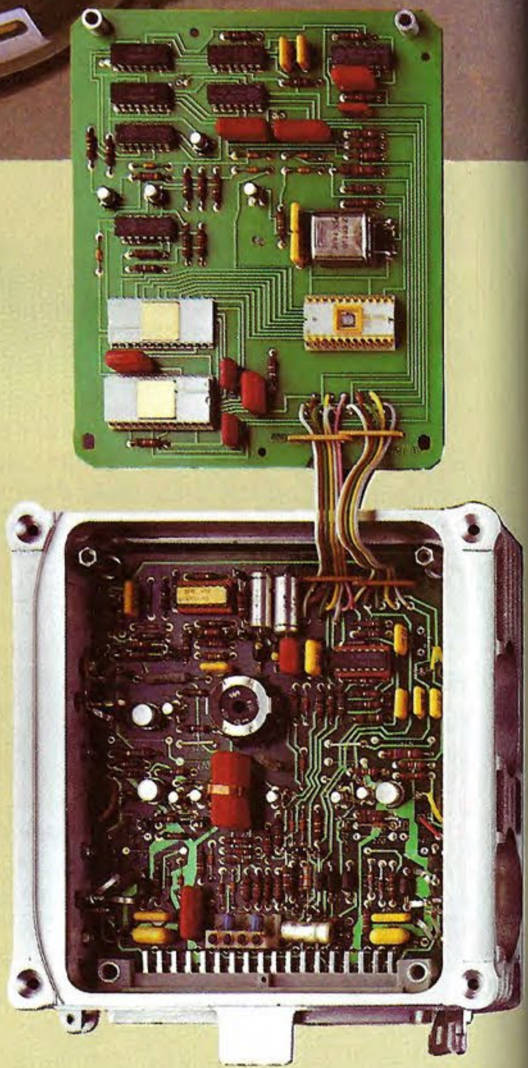
Ahora que los microprocesadores están adquiriendo una importancia creciente en el control de funciones concretas (inyección del combustible, emisión de los gases de escape, encendido, transmisión, frenos, suspensión e instrumentación), ha llegado el momento de ver las posibilidades que tendrían estos sistemas aislados si estuvieran dirigidos por un ordenador central en un coche totalmente automatizado.

Aunque aún no se haya construido un automóvil completamente electrónico, hay algunos indicios que hacen que la idea no resulte tan absurda como podría parecer hace algún tiempo. Hace unos años la empresa alemana Bosch introdujo su sistema de control Motronic para la inyección del combustible y el encendido, que fue instalado en el BMW 732i. La importancia del Motronic es que posee un único microprocesador para el encendido y para la inyección del combustible, y puede además realizar otras funciones, estando provisto de interface y de una programación suplementaria.

En el pasado, la electrónica automovilística tendía a utilizar microprocesadores simples especialmente construidos para realizar funciones concretas. El VLSI, por el contrario, aunque caro, es mucho más versátil y abre el camino de la centralización total; con un microprocesador multiforme como este, la memoria puede programarse para actuar como distribuidor del encendido, para cambiar las marchas, para encender automáticamente los faros o para realizar muchas otras funciones: un solo VSLI puede cumplir las mismas tareas que toda una serie de microprocesadores específicos.

Con todas sus funciones bajo el control de un ordenador central, es mucho más fácil para un automóvil reaccionar a los estímulos externos, como el tráfico y las desviaciones de recorrido. Otra posibilidad consistiría en incorporar un radar de microondas conectado con el ordena-





Proyecto de salpicadero con cuadro indicador de estado sólido y visualizador por reflexión: los datos son proyectados en el parabrisas. A la derecha, el ordenador de a bordo.

dor, el cual determinaría la posición y la velocidad de los demás vehículos y de los peatones, incluso en caso de niebla densa, y activaría automáticamente el sistema de frenado del automóvil para mantenerlo a una distancia segura o, llegado el caso, detenerlo; el radar, además, podría informar al conductor sobre la posibilidad de efectuar un adelantamiento en rutas sinuosas, permitiendo «ver» más allá de las curvas. A finales de los años setenta, sobre todo en la República Federal de Alemania, se experimentaron numerosos sistemas capaces de recoger del exterior los datos relativos a las condiciones del tráfico y a los recorridos alternativos para, con ayuda de un ordenador central, visualizarlos en un panel dentro del automóvil. No hay motivo para que esto no pueda lograrse, puesto que un sistema tal suministraría datos mucho más detallados y, tal vez, la posibilidad de poner bajo control el propio ordenador central en las áreas urbanas de mayor tráfico.

Smiths Industries Ltd.

toma N caracteres de la cadena C$ a partir de la izquierda y los une con la cadena D$. En el gráfico de esta página y la siguiente se muestran, respectivamente, el diagrama y el listado de un programa que ilustra el uso de esta función. A continuación se dan dos aplicaciones de uso general, la primera para las funciones numéricas y la segunda para las cadenas.

Funciones numéricas: solución de ecuaciones de segundo grado. Una ecuación de segundo grado tiene la forma:

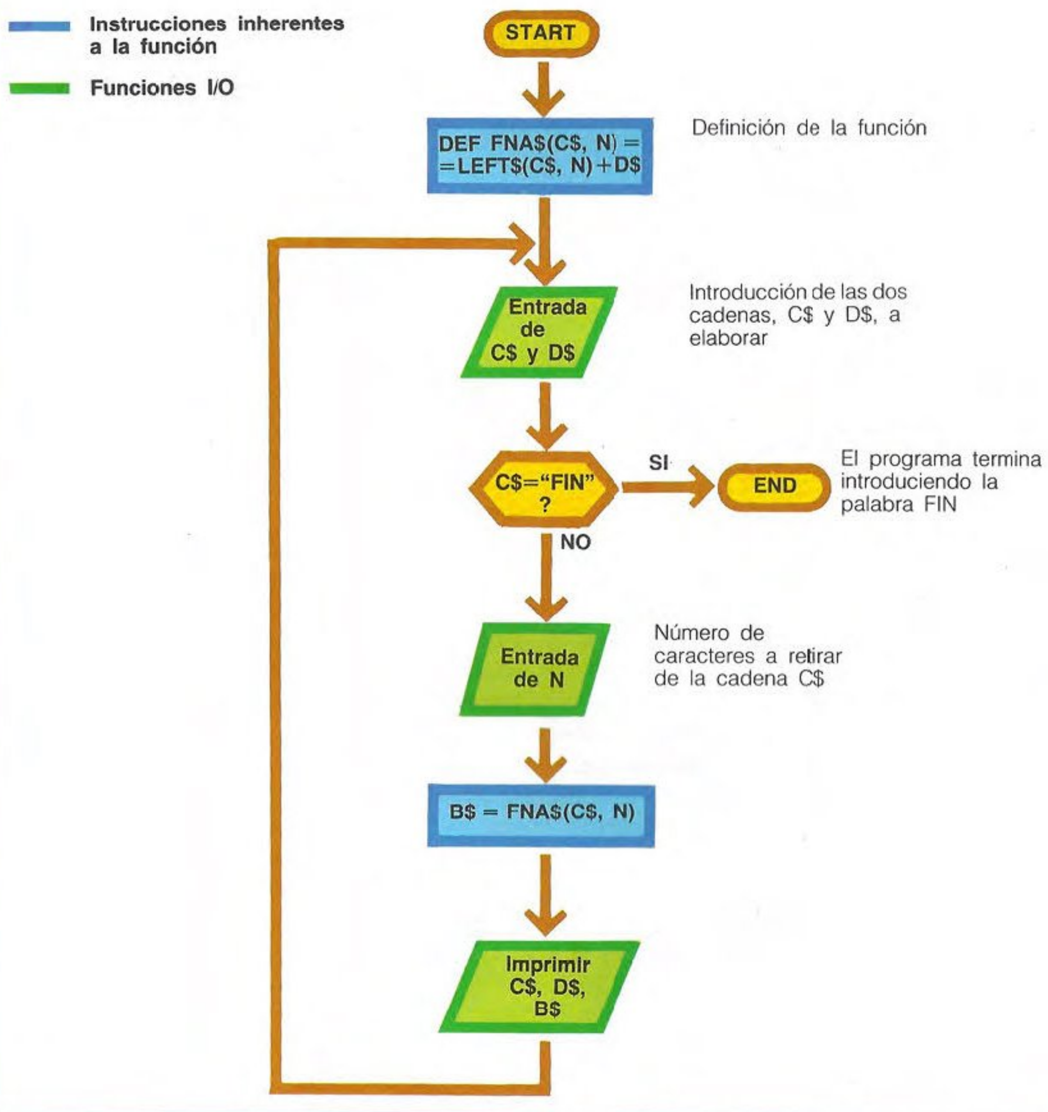
$$A \times X^2 + B \times X + C = 0$$

La solución consiste en obtener, dados los valores de los coeficientes A, B, C, los valores de X que anulan la expresión. Éstos (en número de dos) se obtienen a partir de las fórmulas:

$$X1 = (-B + \sqrt{B^2 - 4 \times A \times C})/(2 \times A)$$
$$X2 = (-B - \sqrt{B^2 - 4 \times A \times C})/(2 \times A)$$

El programa para el cálculo de las soluciones se reduce a la definición y el uso de dos funciones,

USO DE UNA FUNCION DE CADENA



PROGRAMA PARA EL USO DE UNA FUNCION DE CADENA

```
10 ' ** PROGRAMA PARA EL USO DE UNA FUNCION DE CADENA **
20 '
30 '                    FILE = FUNSTR
40 '
50 DEF FNA$ (C$, N)=LEFT$(C$, N)+D$
60 PRINT "INTRODUCIR LAS DOS CADENAS (C$ Y D$) A ELABORAR"
70 PRINT
80 PRINT "PARA TERMINAR INTRODUCIR LA PALABRA (FIN)"
90 INPUT "C$ = ";C$
100 INPUT "D$ = ";D$
110 IF C$="FIN" GOTO 280
120 PRINT "INSERTAR EL NUMERO DE CARACTERES A TOMAR DE C$"
130 INPUT N
140 B$=FNA$(C$, N)
150 ' La línea 140 memoriza en la variable B$ el resultado
160 ' de la función A$(C$,N)
170 '
180 '    INSTRUCCIONES DE IMPRESION
190 '
200 LPRINT "CADENAS A ELABORAR (C$ Y D$)"
210 LPRINT C$
220 LPRINT D$
225 LPRINT "NUMERO DE CARACTERES TOMADOS DE C$ = ";N
230 LPRINT
240 LPRINT "CADENA ELABORADA"
250 LPRINT B$
260 LPRINT
270 GOTO 60        ' RETORNO A UN NUEVO CALCULO
280 END
```

```
CADENAS A ELABORAR (C$ Y D$)
JEFE ALMACEN
URICIO
NUMERO DE CARACTERES TOMADOS DE C$ = 2

CADENA ELABORADA
MAURICIO
```

una para el cálculo de X1, la otra para el cálculo de X2:

DEF FNX1(A,B,C) = (− B + SQR(B∧2 − 4∗A∗C))/(2∗A)
DEF FNX2(A,B,C) = (− B − SQR(B∧2 − 4∗A∗C))/(2∗A)

La única dificultad consiste en controlar la validez de los coeficientes A,B,C. Ciertos valores de estos coeficientes pueden excluir la existencia de soluciones (en el campo de los números reales) o hacer que coincidan (sean iguales numéricamente) las dos soluciones X1 y X2, generalmente distintas. Estas posibilidades vienen dadas por el valor que se obtiene al realizar el cálculo situado bajo el símbolo de raíz cuadrada (este resultado se llama **discriminante**):

Discriminante = $B^2 - 4\times A \times C$ = B∧2 − 4∗A∗C (en Basic)

Según el resultado de este cálculo, tenemos tres casos:

primer caso: el resultado es positivo; hay, entonces, dos soluciones distintas;

segundo caso: el resultado es cero; las dos soluciones coinciden (ambas representadas por un único valor numérico);

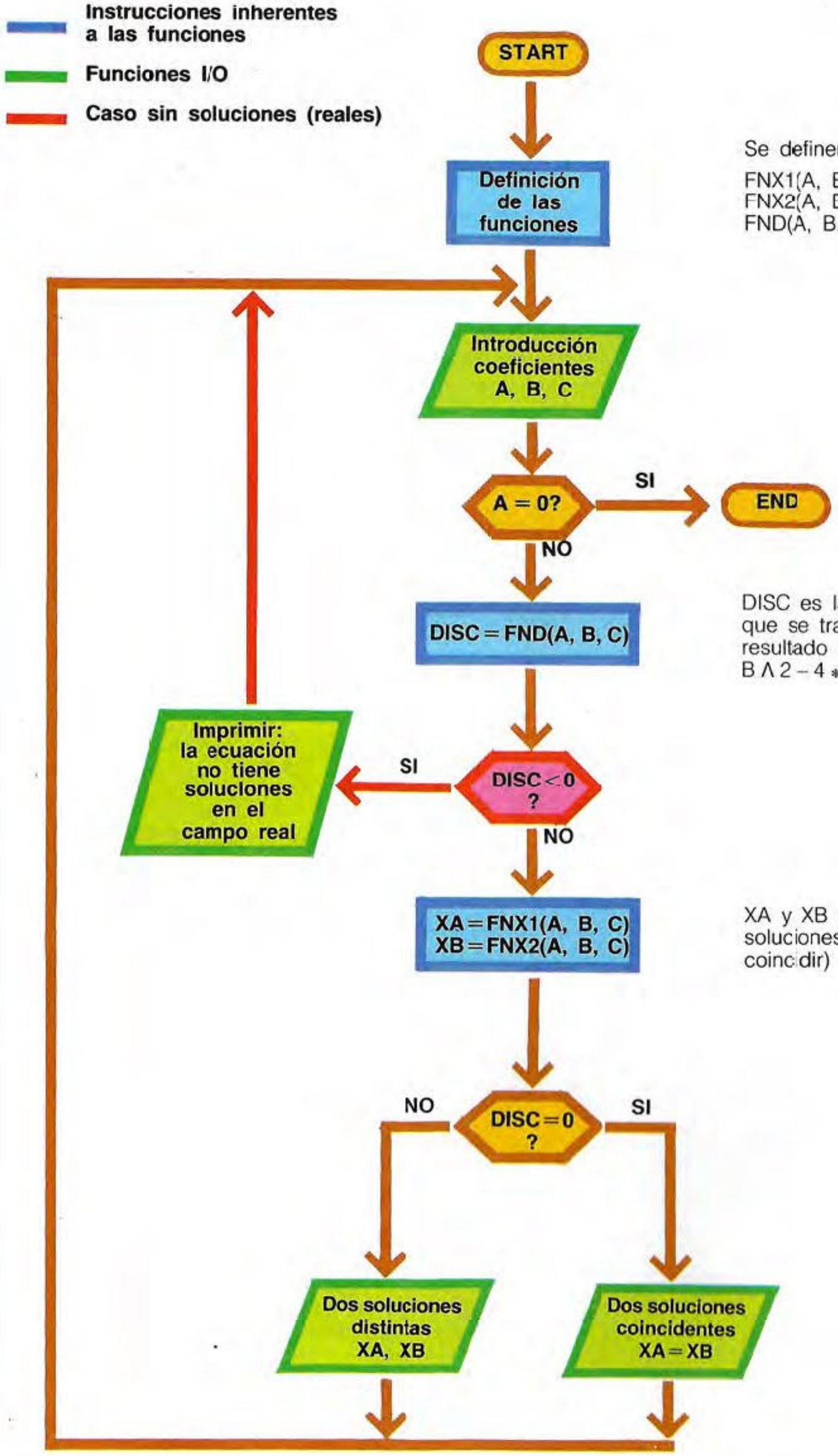
tercer caso: el resultado es negativo; no hay soluciones reales.

Por lo tanto, antes de llamar las funciones para obtener las soluciones de la ecuación, hay que comprobar su existencia, realizando el cálculo anterior (discriminante) y comprobando a qué caso corresponde. Este cálculo también puede ser realizado con una función:

DEF FND(A,B,C) = B∧2 − 4∗A∗C

En la pág. 483 aparece el diagrama del programa, y en la 484 el listado correspondiente.

SOLUCION DE ECUACIONES DE SEGUNDO GRADO



Se definen las funciones:

FNX1(A, B, C), primera solución
FNX2(A, B, C), segunda solución
FND(A, B, C), discriminante

DISC es la variable a la que se transfiere el resultado del cálculo B ∧ 2 − 4 * A * C

XA y XB son las dos soluciones (que pueden coincidir)

SOLUCION DE ECUACIONES DE SEGUNDO GRADO

```
10 ' ** SOLUCION DE ECUACIONES DE SEGUNDO GRADO **
20 '
30 '                FILE = SOLECU
40 '
50 '    FUNCIONES DEFINIDAS POR EL USUARIO
60 DEF FNX1 (A, B, C)=(-B+(SQR(B*B-4*A*C)))/(2*A)
70 DEF FNX2 (A, B, C)=(-B-(SQR(B*B-4*A*C)))/(2*A)
80 DEF FND (A, B, C)=B*B-4*A*C
90 '
100 PRINT "INTRODUCIR LOS COEFICIENTES DE LA ECUACION DE SEGUNDO GRADO (A, B, C)"
110 INPUT "A = ";A
120 INPUT "B = ";B
130 INPUT "C = ";C
140 '
150 IF A=0 GOTO END
160 DISC=FND (A, B, C)         ' A la variable DISC se transfiere el resultado
170 '                            de la función FND (A, B, C), que es la que calcula
180 '                            el DISCRIMINANTE o sea B*B-4*A*C
190 IF DISC<0 THEN GOTO 451
200 XA=FNX1(A, B, C)         ' XA Y XB SON LAS DOS SOLUCIONES
210 XB=FNX2(A, B, C)
220 IF DISC=0 GOTO 250 ELSE GOTO 400
230 '
240 '
250 ' PRIMER CASO: DISCRIMINANTE=0 HAY DOS RAICES REALES Y COINCIDENTES
260 '
270 LPRINT "DISCRIMINANTE=0 : HAY DOS RAICES REALES Y COINCIDENTES XA=XB=";XA
280 GOTO 100               ' RETORNO A UN NUEVO CALCULO
290 '
300 '
400 ' SEGUNDO CASO: DISCRIMINANTE POSITIVO, HAY DOS RAICES REALES Y DISTINTAS
410 '
415 LPRINT "DISCRIMINANTE =";DISC
420 LPRINT "DISCRIMINANTE POSITIVO: HAY DOS RAICES REALES Y DISTINTAS"
422 LPRINT "COEFICIENTES DE LA ECUACION A-B-C"
424 LPRINT "A ="; A, "B ="; B, "C =";C
426 LPRINT
428 LPRINT "RAICES"
430 LPRINT "XA =";XA
440 LPRINT "XB =";XB
445 LPRINT
450 GOTO 100             ' RETORNO A UN NUEVO CALCULO
451 LPRINT "DISCRIMINANTE =";DISC
452 LPRINT "  LA ECUACION NO TIENE SOLUCIONES EN EL CAMPO REAL"
453 LPRINT "PRESENTA COMO SOLUCIONES DOS NUMEROS COMPLEJOS CONJUGADOS"
454 LPRINT
455 LPRINT "COEFICIENTES DE LA ECUACION A-B-C"
456 LPRINT "A =";A, "B =";B, "C =";C:LPRINT:LPRINT
457 GOTO 100
460 END
```

```
DISCRIMINANTE =-471
LA ECUACION NO TIENE SOLUCIONES EN EL CAMPO REAL
PRESENTA COMO SOLUCIONES DOS NUMEROS COMPLEJOS CONJUGADOS

COEFICIENTES DE LA ECUACION A-B-C
A = 2          B = 3          C = 60


DISCRIMINANTE = 88
DISCRIMINANTE POSITIVO: HAY DOS RAICES REALES Y DISTINTAS
COEFICIENTES DE LA ECUACION A-B-C
A = 3          B =-10         C = 1

RAICES
XA = 3.23014
XB = .103195
```

Funciones de cadena: control de mayúsculas y minúsculas. En los programas que requieren la introducción de datos alfabéticos mediante teclado, puede ser necesario introducir en la misma palabra algunas letras en mayúscula y otras en minúscula. Por ejemplo, tras el punto que termina una frase, o al escribir un nombre propio, la primera letra ha de ser mayúscula y las demás minúsculas. Estos cambios continuos son incómodos y pueden causar errores; siempre que sea posible conviene utilizar el propio ordenador para la transformación de la palabra introducida. En la pág. 486 se muestra el diagrama de un procedimiento que convierte en mayúscula la primera letra de un nombre y en minúscula todas las demás. En la pág. 487 se da el listado correspondiente. Las funciones utilizadas son:

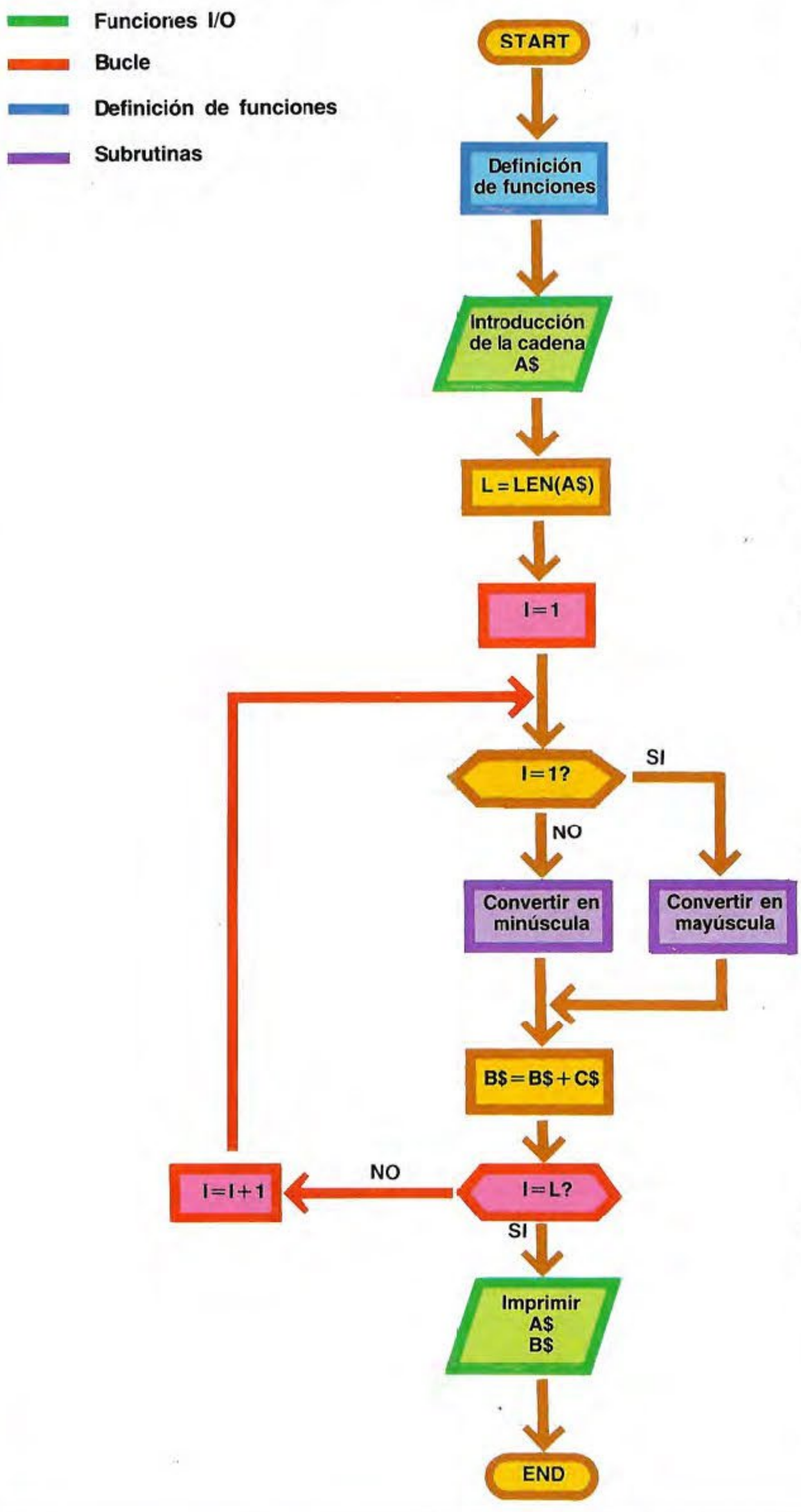
FNP$(I) = MID$(A$,I,1)
(extrae de la cadena A$ el carácter en la posición I)

Estudiar y dar clase con ayuda de un ordenador ya es una costumbre muy difundida en Estados Unidos. En la fotografía, estudiantes de la Georgetown University.



J. Pickerell/Marka

PROGRAMA DE CONTROL Y CONVERSION DE CARACTERES



A$ contiene el nombre en el que hay que efectuar las conversiones de caracteres

La variable L contiene la longitud de la cadena introducida

Puntero al primer carácter de A$

Sólo el primer carácter (I = 1) se convierte en mayúscula. Cambiando el valor en la decisión se puede convertir otro carácter cualquiera

En B$ se acumulan los caracteres convertidos por las subrutinas anteriores

La cadena A$ ha sido convertida por completo y transferida a B$

PROGRAMA DE CONTROL Y CONVERSION DE CARACTERES

```
10 '
20 ' ** PROGRAMA DE CONTROL Y CONVERSION DE CARACTERES **
30 '
40 '                    FILE = PROCON
50 '
60 '    FUNCIONES DEFINIDAS POR EL USUARIO
70 DEF FNP$(I)=MID$(A$, I, 1)         ' EXTRAE DE LA CADENA A$ UN CARACTER EN LA
80 '                                    POSICION (I)
90 DEF FNV$(V)=CHR$(V-32)             ' CONVIERTE LA MINUSCULA EN MAYUSCULA
100 DEF FNZ$ (V) =CHR$ (V+32)         ' CONVIERTE LA MAYUSCULA EN MINUSCULA
110 '
120 INPUT "INTRODUCIR LA CADENA A ELABORAR      ";A$
130 L=LEN(A$)            'LA VARIABLE (L) CONTIENE LA LONGITUD DE LA CADENA A$
140 FOR I=1 TO L
150 IF I=1 THEN GOSUB 1000 ELSE GOSUB 2000
160 '
170 '  LA 1000 Y LA 2000 SON RESPECTIVAMENTE LAS SUBRUTINAS QUE CONVIERTEN
180 '  EN MAYUSCULAS Y EN MINUSCULAS
190 '
200 B$=B$+C$              ' EN B$ SE ACUMULAN LOS CARACTERES CONVERTIDOS POR LAS
210 '                       SUBRUTINAS ANTERIORES
220 NEXT I
230 '
240 '    INSTRUCCIONES DE IMPRESION
250 LPRINT "CADENA ORIGINARIA: A$ =";A$
260 LPRINT "CADENA ELABORADA : B$ =";B$
270 LPRINT                             ESPACIADO
280 END
1000 ' SUBRUTINA DE CONVERSION EN MAYUSCULA
1010 '
1020 C$=FNP$(I)       'Memoriza en C$ el carácter en la posición I
1030 '                 de la cadena A$.
1040 V=ASC(C$)        'Memoriza en V el correspondiente valor numérico
1050 '                 del carácter C$
1060 IF V=>65 AND V=<90  THEN RETURN 'El carácter ya está en mayúscula,
1062 '                                 no hace falta convertirlo.
1070 IF V=>97 AND V=<122 THEN C$=FNV$(V)
1080 RETURN          ' Si V no está comprendido entre 97 y 122 no es una letra
1090 '                 y por tanto no puede ser convertido.
2000 ' SUBRUTINA DE CONVERSION EN MINUSCULA
2010 C$=FNP$(I)
2020 V=ASC(C$)
2030 IF V=>97 AND V=<122 THEN RETURN     ' El carácter ya está en minúscula,
2040 '                                      no hace falta convertirlo.
2050 IF V=>65 AND V=<90 THEN C$=FNZ$(V)
2060 RETURN
```

```
CADENA ORIGINARIA: A$ =PLAZA
CADENA ELABORADA : B$ =Plaza

CADENA ORIGINARIA: A$ =MAYOR
CADENA ELABORADA : B$ =Mayor

CADENA ORIGINARIA: A$ =CALLE
CADENA ELABORADA : B$ =Calle

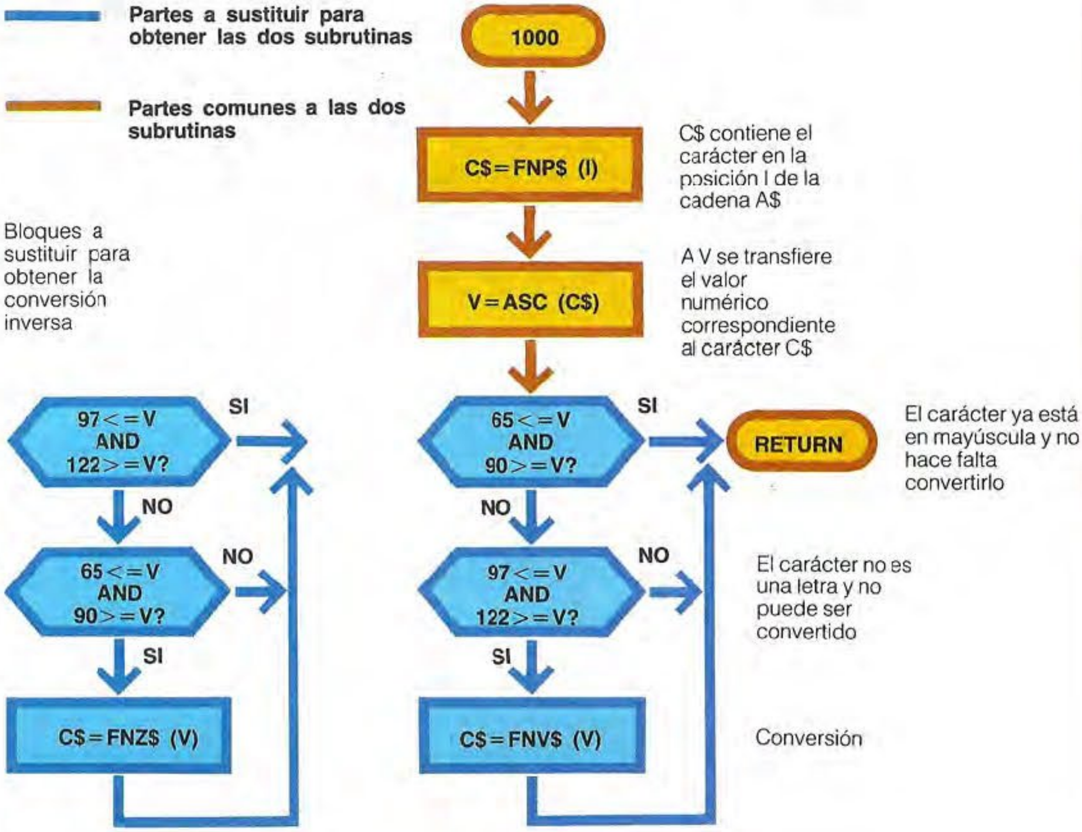
CADENA ORIGINARIA: A$ =GOYA
CADENA ELABORADA : B$ =Goya
```

FNV$(V) = CHR$(V − 32)
(conversión de minúscula en mayúscula)
FNZ$(V) = CHR$(V + 32)
(conversión de mayúscula en minúscula)

(Ver en la pág. 488 la subrutina de la conversión de minúscula en mayúscula y la subrutina inversa con las variaciones a introducir. Arriba aparece el listado, más detallado).

SUBRUTINAS DE CONVERSION EN MAYUSCULA

Entrada A$=Cadena que contiene el carácter a convertir en la posición I



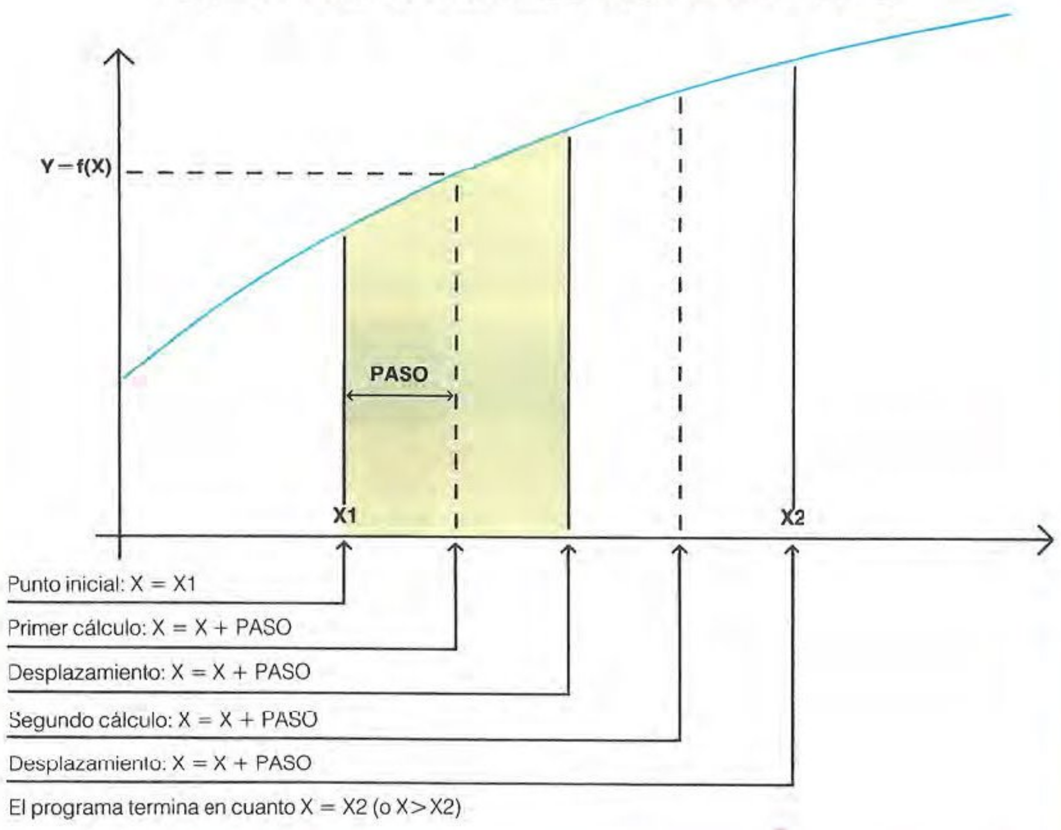
Aplicaciones concretas. La utilización de funciones definidas por el usuario es especialmente útil en dos campos muy definidos: en las aplicaciones científicas y en algunos programas o subrutinas que prevén el uso intensivo de datos introducidos mediante teclado, por ejemplo para la generación de archivos. Dada la complejidad del tema, esta segunda utilización, que incluye la gestión de la pantalla para la creación de «máscaras», se verá por separado en el capítulo dedicado a la unidad de vídeo.

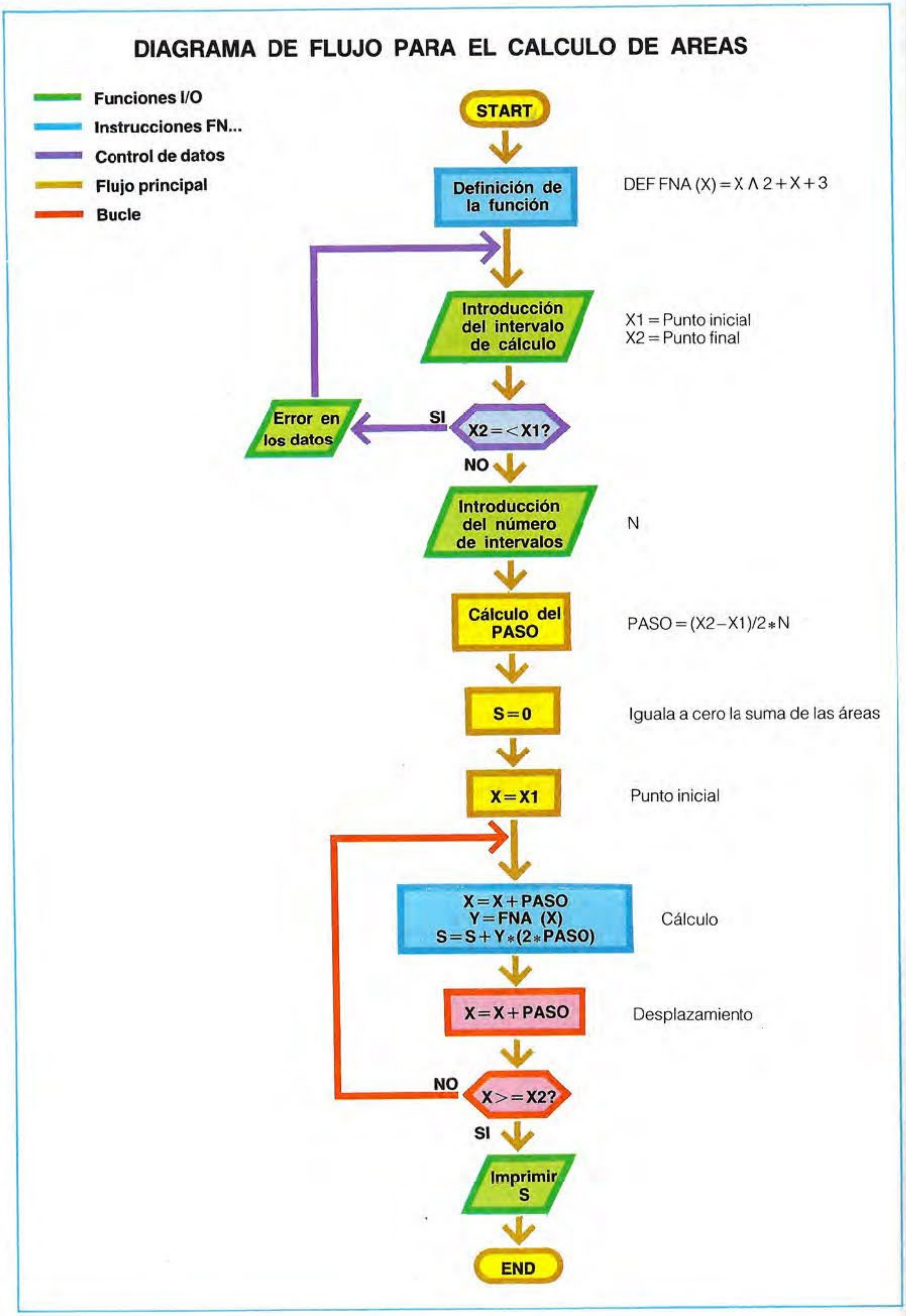
A continuación se muestra una aplicación de las funciones numéricas para el cálculo de áreas. El método expuesto no es riguroso, pero ofrece una validez aceptable y proporciona un ejemplo, el más sencillo, de aplicación del ordenador a los problemas matemáticos de ingeniería.

Anteriormente (ver págs. 333 y 334), el mismo problema fue resuelto por un método gráfico. El

usuario tenía que dividir la figura en un cierto número de zonas y dar al ordenador las coordenadas de compensación; el programa se limitaba a sumar las áreas elementales. Utilizando las funciones, todo el trabajo puede ser realizado por la máquina; el usuario sólo ha de suministrar los límites entre los que desea realizar el cálculo y el número de intervalos en que ha de dividirse el intervalo total. Abajo vemos un gráfico que ilustra el significado de los diversos parámetros; en la pág. 490, el diagrama de flujo y, en la 491, el listado del programa, con algunas salidas obtenidas dividiendo el mismo intervalo (5, 25) en distinto número de partes. Al aumentar las divisiones aumenta la precisión del cálculo, hasta un límite (alrededor de 80) más allá del cual no hay variaciones significativas en el resultado.
La elección de dicho límite es muy difícil de efectuar a priori. La práctica normal consiste en efectuar algunas pruebas para determinar la marcha del proceso, en base a la cual poder elegir, aproximadamente, el valor óptimo que

DIAGRAMA ILUSTRATIVO DEL CALCULO DE AREAS




DIAGRAMA DE FLUJO PARA EL CALCULO DE AREAS
Funciones I/O
Instrucciones FN...
Control de datos
Flujo principal
Bucle
START
Definición de la función
DEF FNA (X) = X ∧ 2 + X + 3
Introducción del intervalo de cálculo
X1 = Punto inicial
X2 = Punto final
Error en los datos
SI
X2 = < X1?
NO
Introducción del número de intervalos
N
Cálculo del PASO
PASO = (X2 − X1)/2 * N
S = 0
Iguala a cero la suma de las áreas
X = X1
Punto inicial
X = X + PASO
Y = FNA (X)
S = S + Y * (2 * PASO)
Cálculo
X = X + PASO
Desplazamiento
NO
X > = X2?
SI
Imprimir S
END

PROGRAMA PARA EL CALCULO DE AREAS ELEMENTALES

```
10 '
20 ' ** PROGRAMA PARA EL CALCULO DE AREAS ELEMENTALES **
30 '
40 '                FILE = CALAR
50 '
60 ' FUNCIONES DEFINIDAS POR EL USUARIO
70 DEF FNA(X)=X*X+X+3      ' La función representa una curva de segundo orden
80 PRINT "INTRODUCIR LOS EXTREMOS ENTRE LOS QUE SE DEFINE LA FUNCION"
90 INPUT "PUNTO INICIAL: X1 = ";X1
100 INPUT "PUNTO FINAL: X2 = ";X2
110 IF X2=<X1 THEN PRINT "ERROR EN LOS DATOS": GOTO 80
120 PRINT "INTRODUCIR EL NUMERO DE PARTES EN QUE SE DESEA DIVIDIR EL AREA"
130 INPUT "N = ";N
140 '  CALCULO DEL PASO
150 '
160 PASO=(X2-X1)/(2*N)
170 S=0          ' PONE A CERO LA SUMA DE LAS AREAS
180 X=X1
190 X=X+PASO
200 Y=FNA(X)
210 S=S+Y*(2*PASO)
220 X=X+PASO
230 IF X=>X2 GOTO 1000 ELSE GOTO 190
1000 '  INSTRUCCIONES DE IMPRESION
1002 LPRINT "NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA =";N
1010 LPRINT "AREA TOTAL =";S
1020 LPRINT
1030 END
```

```
NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA = 10
AREA TOTAL = 5520

NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA = 20
AREA TOTAL = 5525

NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA = 40
AREA TOTAL = 5526.25

NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA = 80
AREA TOTAL = 5526.56

NUMERO DE PARTES EN QUE SE HA DIVIDIDO EL AREA = 800
AREA TOTAL = 5526.69
```

supone el mejor compromiso entre precisión y tiempos de ejecución. Como alternativa, se puede usar la versión compilada del programa que, gracias a sus altas velocidades de ejecución, minimiza el problema del tiempo y permite efectuar un gran número de subdivisiones.

Ejemplo de aplicación: calefacción de ambientes

El uso de subrutinas es esencial en problemas de cierta complejidad. Un programa único que contenga todas las funciones a realizar es difícil de plantear, puesto que hay que ir en seguida a los detalles. Por el contrario, el uso de subrutinas permite trabajar con distintos grados de detalle, diluyendo las dificultades en diferentes puntos, cada uno con una función propia limitada y definida. Al principio se analiza el problema en su conjunto y se diseña el diagrama general; luego se desarrollan las funciones detalladas (subrutinas) en diagramas aparte.

La mejor forma que se puede dar a un programa consiste en definir un cuerpo principal (main program, o simplemente MAIN), cuya única tarea consiste en llamar (y conectar) una serie de subrutinas. La estructura misma del programa (main y subrutinas) sugiere el método de análisis. En un primer tiempo se analiza el problema globalmente, con lo que se obtiene el diagrama

...Y como maestro, la Tortuga

El papel que la informática puede jugar en los procesos de aprendizaje es un tema de interés ampliamente reconocido, tanto en el campo de la enseñanza como en el de la informática.
Los ordenadores se utilizan ya en el marco de actividades docentes y de entrenamiento en empresas y grandes organismos administrativos, universidades y centros de investigación, laboratorios de investigación pedagógica e institutos de enseñanza. En las últimas décadas se han proyectado, experimentado y en gran medida construido y vendido a escala industrial sistemas basados en el papel del ordenador en un amplio campo de aplicaciones.

- *Sistemas que utilizan el ordenador como instrumento para enseñar o como auxiliar didáctico en ambientes de tipo tradicional (actividad de formación dirigida, tanto de tipo universitario o escolar como empresarial). Estos sistemas se designan normalmente con el nombre de Instrucción Asistida por Ordenador (IAO) y son el resultado de las investigaciones e inversiones realizadas a partir de la segunda mitad de los años cincuenta, sobre todo en las universidades estadounidenses. En el marco de las actividades IAO se han empleado sistemas del tipo: instrucción programada, adiestramiento y entrenamiento, resolución de problemas, simulación, juegos educativos, tests.*
- *Sistemas de integración de distintos medios, que utilizan, además del ordenador, medios audiovisuales y televisivos, bancos de datos, teleconexiones. Sistemas de este tipo se utilizan sobre todo en actividades de instrucción a distancia, con el apoyo, en muchos casos, de programas difundidos por las redes de televisión.*
- *Sistemas «inteligentes», basados en las investigaciones realizadas en el campo de la inteligencia artificial, que gestionan una interacción «abierta» con el estudiante, sobre la base de lenguajes elementales y creativos, que valoran la aprehensión de capacidades lógicas mediante la elaboración y corrección de programas por el ordenador. Estos sistemas se han empleado en el campo de la educación infantil y en el de la recuperación de disminuidos psíquicos.*

Sobre todo a partir de los años setenta, la difusión de los ordenadores en escuelas de todo tipo y grado ha adquirido amplias dimensiones en los países más industrializados, en los que se han seguido políticas nacionales de financiación y experimentación de dichos sistemas. En Estados Unidos, el gobierno federal ha invertido, principalmente a través de la National Science Foundation, *unos 300 millones de dólares hasta finales de los sesenta, tras lo cual la responsabilidad ha sido asumida por el Estado.*
En Japón, Gran Bretaña, Alemania y Francia se han seguido políticas nacionales de experimentación a partir de principios de los setenta, sobre todo bajo la influencia del Centro de Investigación sobre la Educación del OCSE. A nivel europeo, la CEE, a través del Fondo Social Europeo, financia proyectos en este campo, orientados sobre todo a la formación profesional. En Italia, las actividades de investigación han sido promovidas por institutos del CNR y de la universidad, por asociaciones educativas, etc.
Hay que señalar, sin embargo, que el interés hacia los sistemas de instrucción basados en los ordenadores parece, actualmente, contemplar fenómenos más complejos y exigencias más profundas que los primeros grandes proyectos de «máquinas para enseñar», en los albores de la historia de la IAO.
A finales de los años cincuenta, cuando los ordenadores entraron en las escuelas y universidades norteamericanas, el ambiente educativo estaba influido por las corrientes pedagógicas y las teorías de la instrucción programada.
En este contexto –de las investigaciones de Patrick Suppes y Richard Atkinson en el Instituto de estudios matemáticos aplicados a las ciencias sociales de la Universidad de Stanford, de Donald Bitzer para el sistema PLATO en los laboratorios de la Universidad de Illinois, y de los laboratorios IBM para el sistema CAI 1500– nace la primera generación de sistemas IAO.
Las experiencias realizadas durante los años sesenta se resienten notablemente de las características tecnológicas de esta primera generación. Las estrategias basadas en los conceptos de la instrucción programada implican rígidos controles del diálogo programado. Sin embargo, ya a partir de la segunda mitad de los años sesenta empieza a desarrollarse una nueva línea teórica y experimental respecto al empleo de ordenadores en los procesos de aprendizaje. Al uso del ordenador como electronic page turner *(«pasador de página» electrónico), es decir, al modelo del IAO como mera automación del tradicional aprendizaje acumulativo y lineal, se opone un nuevo concepto, cuyo lema*

es «dejemos que sean los niños quienes programen los ordenadores, y no los ordenadores los que programen a los niños». Estas palabras son de Seymour Papert, un experto en cibernética que, tras haber trabajado varios años junto al psicólogo suizo Jean Piaget, ha asumido, junto con Marvin Minsky, la dirección del Laboratorio de Inteligencia Artificial del MIT.
El niño no aprende del ordenador, sino programando el ordenador, que lo pone en condiciones de explicitar sus «intuiciones». Traducir una intuición en forma de programa significa concretarla, volverla más sujeta a reflexión.
Este enfoque –como es evidente por otra parte por la biografía de Papert– se basa en la integración de los modelos de las ciencias cognoscitivas con las investigaciones realizadas en el campo de la inteligencia artificial, y se desarrolla en ambientes experimentales mediante el empleo de lenguajes altamente interactivos.
Históricamente se puede decir que el Basic ha sido el primer lenguaje que hizo posible un aprendizaje basado en la programación. Sin embargo, lo que el Basic permite en los cálculos matemáticos, nuevos lenguajes interactivos lo permiten en el campo no numérico. El LOGO y el SMALLTALK, concretamente, desarrollan capacidades de manipulación de objetos simbólicos, gráficos y textuales.
Cada lenguaje está dotado de un conjunto de comandos básicos. Con una adecuada secuencia de dichos comandos se puede escribir un programa. La característica de un lenguaje como el LOGO es que permite procedimientos y programas dotados de un nombre y de uno o más temas que, una vez definidos, pueden ser «llamados» de forma no distinguible (para quien escribe el programa) de los comandos primitivos. En otros términos, un lenguaje como el LOGO permite aumentar el número de palabras a las que dar un significado operativo.
De esta forma, es posible describir una operación compleja como si fuese simple mediante una serie de operaciones de complejidad menor. Esta manera de «descomponer» los problemas por niveles de complejidad decreciente es, bien mirado, el arte fundamental de la investigación científica, el más importante de los métodos heurísticos.
En efecto, conocer significa convertir lo desconocido en conocido mediante estructuras lógicas controlables en cada paso de la descomposición. Este procedimiento es válido para el conocimiento científico y para el aprendizaje. Al mismo tiempo, la descomposición sistemática de una acción compleja es la habilidad fundamental necesaria para proyectar acciones destinadas a alcanzar objetivos no inmediatos.
El SMALLTALK es un lenguaje desarrollado por Alan Kay en el Centro de Investigaciones Xerox de Palo Alto, orientado hacia la creación de ambientes para la programación, que ha sido experimentado con fines didácticos y se presta a la programación para dibujos animados, juegos y composiciones musicales.
El SMALLTALK se basa en una visión del proceso intelectivo como concurrencia y cooperación de procedimientos. Si, por ejemplo, el objetivo es componer un dibujo con dos tipos de simetría fundamentales, se puede crear un procedimiento que realiza el primer tipo de simetría, ponerlo en marcha y pensar en el segundo. Cuando el segundo procedimiento ya está escrito, se puede retomar la ejecución del primero y ver qué resultados ha dado. Mientras tanto, el segundo puede ser puesto en ejecución. Finalmente, los dos procedimientos tendrán que ser ejecutados en paralelo para obtener el producto final, o sea el dibujo previsto. Es evidente que la programación paralela facilita la programación de sistemas complejos y la vuelve más clara.
La utilización de lenguajes interactivos en ambientes experimentales, sobre todo con niños, ha sido realizada por Papert y Minsky inspirándose en la filosofía piagetiana de la inteligencia como adaptación al ambiente y de la educación como desarrollo espontáneo a través de una serie casi biológica de etapas intelectuales.
De esta premisa nace la idea de que, en este procedimiento, hay que situar el sujeto del aprendizaje (normalmente el niño) en un ambiente fecundo. Surge así el Laboratorio de Inteligencia Artificial del MIT, y en su seno un laboratorio experimental sobre el aprendizaje basado en ordenadores, mediante un lenguaje de programación simple: el LOGO.
En el laboratorio del MIT han «jugado» personas de todo tipo. Desde niños de cuatro años hasta clases enteras de chicos procedentes de zonas culturalmente bajas de los alrededores de Boston; desde estudiantes superdotados hasta los especialistas en ordenadores del MIT; desde investigadores de algunas universidades de Latinoamérica, donde el LOGO ha sido trasplantado, hasta los de otros departamentos de Inteligencia Artificial, como el de Edimburgo. Con el

LOGO y su filosofía de aprendizaje se han formado muchos de los más brillantes investigadores del Laboratorio de Inteligencia Artificial. Los resultados de este trabajo se guardan en más de doscientos informes técnicos, que abordan problemas de los más diversos tipos, desde la creación automática de poemas a la simulación de fenómenos biológicos y de comportamiento.

El lenguaje LOGO es un sistema para dialogar con el ordenador, y permite dar instrucciones descomponiento un determinado proceso en una serie de acciones lógicas. En otros términos, es un auténtico autómata, es decir, un fiel ejecutor de órdenes correctamente formuladas.

El autómata LOGO no ejecuta órdenes cualesquiera. Al principio reconoce un conjunto de palabras originarias mediante las cuales se pueden componer las órdenes. Pero el LOGO es un autómata capaz de aprender: su lenguaje puede ser ampliado, en el sentido de que mediante las palabras originarias el usuario puede definir el significado operativo de otras palabras, que pasan así a formar parte de las órdenes que el autómata puede ejecutar. Partiendo de alguna de las materias enseñadas en las escuelas (lengua, matemáticas, ciencias, música), las experiencias realizadas con el LOGO han rebasado pronto los límites de estas disciplinas, proponiendo a los niños experiencias de tipo global, no realizables sino con el ordenador.

Entre estas experiencias, la más conocida, implantada también en otros lenguajes (por ejemplo, el SMALLTALK, o el LISP, que es el lenguaje básico de la inteligencia artificial) es la denominada «de la Tortuga». La Tortuga es un pequeño vehículo, un auténtico robot teledirigido por el ordenador mediante el lenguaje LOGO. El objetivo es explorar un espacio mediante la Tortuga, que se desplaza por el terreno dejando un rastro sobre el suelo, y que puede detectar la presencia de obstáculos en su camino mediante células fotoeléctricas.

La Tortuga ha sido también «transferida» a la pantalla de un ordenador, donde aparece como un «objeto», un punto o un triangulito. Sus desplazamientos permiten trazar cualquier figura, gracias a sencillísimas órdenes dadas mediante el teclado: ADELANTE, ATRAS, DERECHA, IZQUIERDA, acompañadas de los valores de distancia lineal y variación de dirección del desplazamiento en grados. Cada instrucción puede así organizarse en secuencias más o menos complejas, que acaban constituyendo auténticos procedimientos, descubiertos y elaborados de forma autónoma, confrontando intuiciones y operaciones lógicas, a fuerza de pruebas y errores.

Las órdenes ADELANTE y ATRAS determinan un desplazamiento de la Tortuga a lo largo de la recta que pasa por su eje. Para modificar su dirección hacen falta otras órdenes:

DERECHA e IZQUIERDA conllevan una rotación in situ de la Tortuga. Todas estas órdenes han de ir acompañadas de un número, en el mensaje de entrada, que le indica a la Tortuga la longitud del desplazamiento lineal o cuánto ha de girar. Para un adulto es evidente que el número, en el caso de la rotación, representa la medida en grados del ángulo de rotación. Pero para la mayoría de los niños, esto constituye materia de investigación.

Para obtener un cuadrado habrá que dar las siguientes órdenes:

```
PARA CUADRADO
ADELANTE 100
DERECHA 90
ADELANTE 100
DERECHA 90
ADELANTE 100
DERECHA 90
ADELANTE 100
DERECHA 90
FIN
```

o bien	*o bien*
PARA CUADRADO REPETIR 4 ADELANTE 100 DERECHA 90 FIN	PARA CUADRADO:N REPETIR 4 ADELANTE:N DERECHA 90 FIN

De la misma manera se puede programar un triángulo equilátero:

PARA TRIANGULO ADELANTE 100 DERECHA 120 ADELANTE 100 DERECHA 120 ADELANTE 100 DERECHA 120 FIN	*o bien* PARA TRIANGULO: N REPETIR 3 ADELANTE: N DERECHA 120 FIN

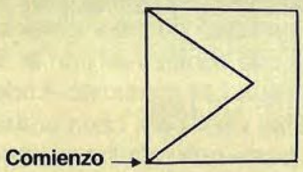
No hace falta un ordenador para dibujar un cuadrado o un triángulo; bastaría con un lápiz y un

papel. Pero estos programas, una vez elaborados, constituyen para el niño elementos de construcción que le permiten establecer una jerarquía de conocimientos. A partir de este juego se desarrolla una agilidad intelectual que podrá emplear después en innumerables circunstancias, como se desprende claramente de los proyectos que el niño emprende tras algunas sesiones de trabajo con la Tortuga.

Un ejemplo entre los más elementales, pero significativo, es el del programa PARA CASA. Tras haberle enseñado al ordenador las palabras CUADRADO y TRIANGULO, se puede intuir que para dibujar una casa bastará con colocar un triángulo sobre un cuadrado. En el primer intento, el programa será:

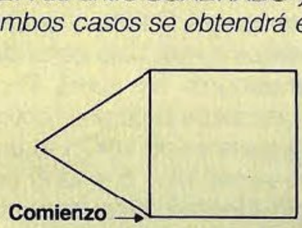
PARA CASA
CUADRADO
TRIANGULO
FIN

Pero la orden así formulada dará este resultado:

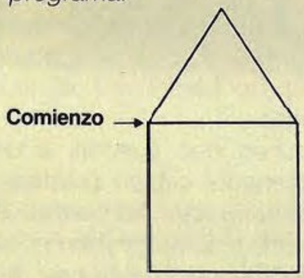


El triángulo ha sido dibujado dentro del cuadrado, en lugar de encima.

Hay distintas maneras de descubrir y corregir el error. Un intento podría ser reconstruir el camino de la Tortuga, descubriendo que el triángulo ha sido dibujado dentro del cuadrado porque se ha pedido a la Tortuga que dibuje el triángulo girando hacia la derecha. Entonces se podrá intentar resolver el problema programando el triángulo con rotaciones orientadas hacia la izquierda. O bien se podrá insertar la instrucción IZQUIERDA 60 entre CUADRADO y TRIANGULO. En ambos casos se obtendrá esta figura:



El aprendiz de programador va progresando. Constata que a menudo las cosas no son ni completamente correctas ni completamente erróneas, y que se puede pasar de un caso al otro sin solución de continuidad. Esta casa es más convincente que la primera, pero todavía no es perfecta, hay que enderezarla. Para resolver este último problema bastará simplemente con dar la orden DERECHA 90 como primer paso del programa.



Además de la Tortuga, las experiencias educativas realizadas mediante el LOGO han sido: 1) la realización de poemas con el ordenador, en base a una reconstrucción «productiva» de la gramática y del concepto de generación aleatoria de palabras de un determinado tipo; 2) el estudio empírico de las regularidades de los números y de las operaciones aritméticas; 3) experiencias de composición musical mediante una caja de música controlada por ordenador con el lenguaje LOGO, que permite descomponer la música en sus elementos fundamentales (melodía, ritmo, armonía, color); a través de la variación, efectuada mediante comandos LOGO, de cada uno de estos elementos, se llega a comprender su importancia y valor efectivo.

El aspecto cognoscitivo e interactivo de la informática que subyace a lenguajes como el LOGO y el SMALLTALK, propone un uso del ordenador como instrumento que permite convertir la actividad de programación en un método para la formalización de los conocimientos y la adquisición de capacidades de comprensión y manipulación de problemas y situaciones complejos. Es algo que va mucho más allá del tradicional concepto de la IAO y que, bien mirado, revela un posible planteamiento de la informática en los ambientes escolares, incluso donde el objetivo no es exclusivamente la obtención de una profesionalidad especializada.

(Extracto de «Informática y procesos cognoscitivos», de B. Zeller, en QUADERNI DI INFORMATICA, AÑO X, n.º 2, 1983, Honeywell Information Systems Italia. El ejemplo está tomado de «Mindstorms», de S. Papert.)

del main; sucesivamente, se desciende al detalle, analizando las funciones que deberá cumplir cada subrutina y diseñando su diagrama de flujo. Estas fases pueden parecer laboriosas y difíciles, acaso poco relacionadas con el empleo de la máquina; pero hay que recordar que el ordenador sólo puede resolver los problemas si previamente ha sido debidamente «instruido». Un proyecto de software demasiado apresurado o incompleto conduce a programas defectuosos y poco funcionales, si no decididamente erróneos.

Las aplicaciones que pueden automatizarse mediante ordenador cubren prácticamente todos los campos de actividad y a menudo el programador ha de resolver problemas relativos a temas que no conoce. En este caso tendrá que ser el usuario final quien suministre los métodos de cálculo, participando activamente en el planteamiento general.

Como ejemplo de análisis de problemas de cierta complejidad, se desarrollan a continuación los diagramas de un programa aplicable al cálculo de la calefacción de ambientes. El ejemplo ha de considerarse como la descripción de una metodología general, más que una aplicación específica. La realización de la tarea se divide en seis partes:

1 / Análisis general. En esta fase se definen las magnitudes fundamentales y las fórmulas de cálculo
2 / Síntesis y preparación del diagrama del main
3 / Desarrollo del análisis relativo a las subrutinas y preparación de los diagramas de cada una de ellas
4 / Escritura de las subrutinas y las pruebas de validez
5 / Escritura del main
6 / Integración del main con las subrutinas y pruebas integradas

Las fases 4, 5 y 6 prevén la escritura de cada subrutina, que tendrá que ser probada por separado antes de proceder a la integración, y las pruebas del programa entero. Más adelante se expondrán otros métodos de comprobación del software.

Análisis general

En esta fase ante todo hay que obtener una visión global del problema específico y de las leyes que lo gobiernan, y es esencial la colaboración de un eventual usuario del programa para la definición de los objetivos y de los algoritmos a utilizar. En este caso concreto hay que conocer las leyes físicas que regulan la calefacción de ambientes.

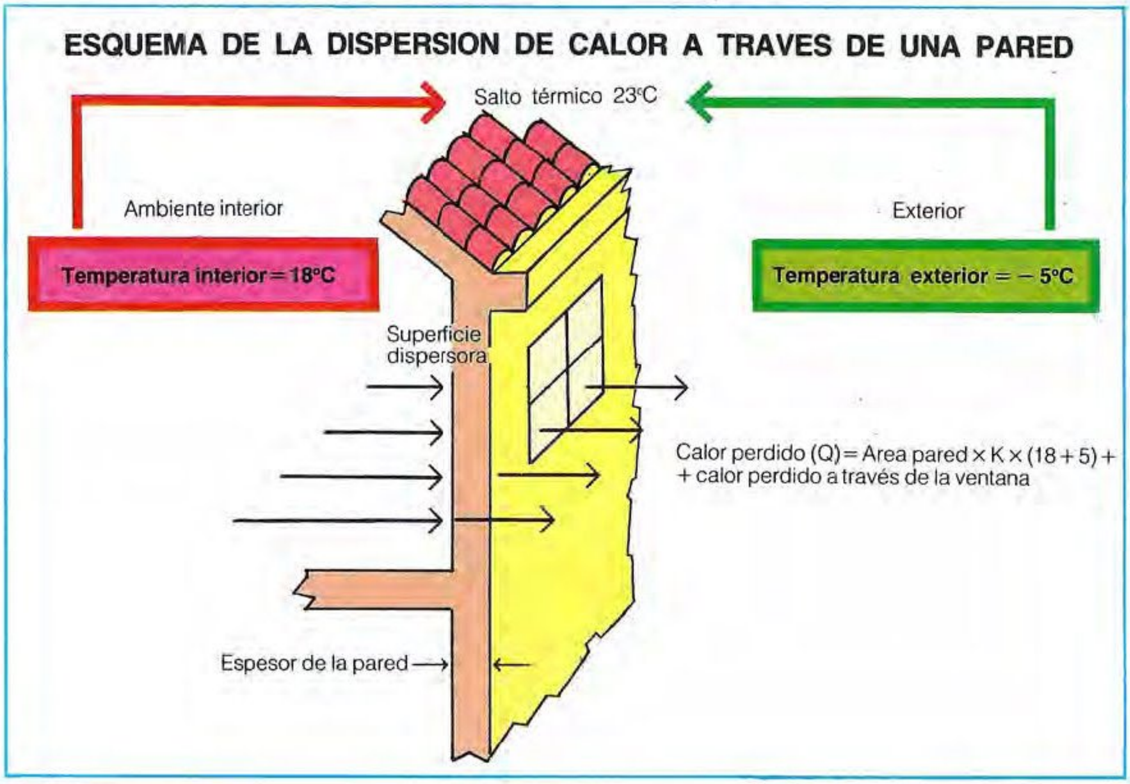
Un ambiente cualquiera, por ejemplo una habitación de una casa, puede considerarse como una «caja» que contiene aire caliente (a la temperatura que se desee), rodeada por el aire frío del ambiente exterior. El calor del interior pasa a través de las paredes y se dispersa por el exterior. De esta forma, la habitación se enfría y pronto la temperatura interna se iguala a la externa. Para mantener constante la temperatura hay que suministrar a la habitación tanto calor como pierde hacia el exterior. Así, si la habitación pierde en una hora 1.250 calorías, habrá que suministrarle otras tantas quemando la correspondiente cantidad de combustible.

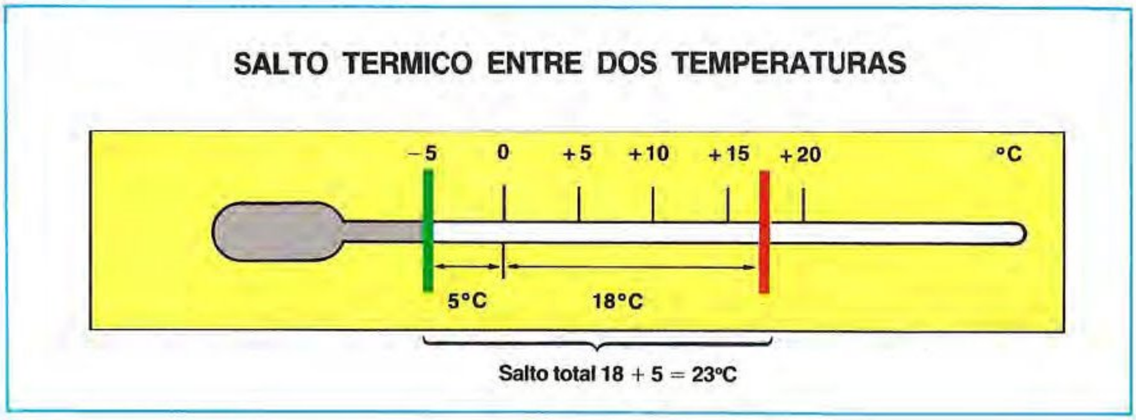
Si conociéramos la cantidad de calor perdida, el problema estaría resuelto y no necesitaríamos programa alguno. Sin embargo, el calor perdido depende de numerosos factores, y el objetivo del programa es precisamente llevar a cabo un cálculo que, teniendo en cuenta esos factores, suministre la cantidad de calor perdida y, por tanto, que hay que compensar con la combustión de una cantidad de combustible adecuada. Conociendo este valor para cada ambiente de nuestra casa, basta con calcular la suma de dichos valores para deducir la potencia, en calorías por hora, que ha de tener la calefacción.

La cantidad de calor que se pierde a través de paredes y ventanas hacia el exterior (ver tabla superior de pág. 498) la da la relación:

$$Q=\begin{pmatrix}\text{Area}\\\text{pared}\end{pmatrix}\times K\times\left[\begin{pmatrix}\text{Temperatura}\\\text{interna}\end{pmatrix}-\begin{pmatrix}\text{Temperatura}\\\text{externa}\end{pmatrix}\right]\quad(1)$$

Q es la cantidad de calor perdida en calorías/hora, y K es un coeficiente que depende del tipo de pared y de su espesor. Obsérvese que el término (Temperatura interna-Temperatura externa) representa el salto de la temperatura entre los dos ambientes (interno y externo): si la temperatura externa está bajo cero, dicha diferencia se convierte en una suma. Por ejemplo (ver gráfico superior de la página contigua), si la temperatura interna es de 18°C y la externa de –5°C, el salto es de 18 + 5 = 23°C (ver el segundo gráfico). Efectivamente, hacen falta 18°C para pasar de la temperatura interna a cero, y otros 5° para pasar de cero a –5°C (temperatura externa). Para realizar los cálculos es necesario





conocer el valor de K. Hay numerosas tablas técnicas que prevén la totalidad de los casos prácticos; según lo completo que deseemos que sea nuestro programa, tendremos que incluir en él un cierto número de valores, de forma que cubran la oportuna gama de casos posibles. En este ejemplo se dan tablas muy simplificadas, pero el método expuesto, que seguidamente se completará con el programa correspondiente, admite ampliaciones. Los valores de K previstos en el programa se dan en la tabla superior de la pág. 498. Antes de abordar la preparación de los diagramas, para compren-

Tipo de superficie	Espesor (centímetros) 25	38	50	Siglas	Tabla en el ordenador
Muro de ladrillo	1.6	1.3	1.0	ML	KL(3)
Muro de cemento	2.2	1.8	1.6	MC	KC(3)
Suelo		1.0		SU	
Techo		2.0		TE	
Puerta		4.0		PU	
Vertana		5.0		VE	

KL(1) = 1.6
KL(2) = 1.3
KL(3) = 1.0

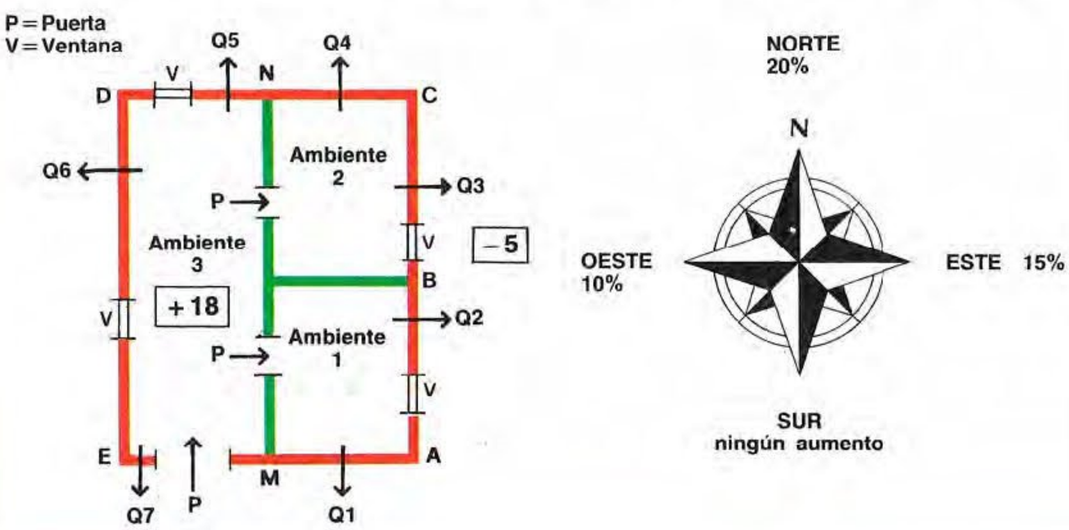
KC(1) = 2.2
KC(2) = 1.8
KC(3) = 1.6

Valores de K para algunas superficies. En la última columna se dan las siglas mediante las cuales la máquina puede reconocer el tipo de superficie.
En las últimas denominaciones (suelo, etc.) no se dan diferencias en función del espesor; en las aplicaciones prácticas las diferencias son marginales, y se puede tomar K como valor medio que cubre, en una primera aproximac ón, todos los casos.

der mejor el mecanismo de cálculo, resolveremos manualmente un caso posible. El programa consistirá en la escritura parametrizada de una serie de instrucciones que harán realizar las mismas operaciones a la máquina.
Consideremos el plano de un apartamento (ver gráfico inferior). Las paredes en verde son las interiores, y separan ambientes que se hallan a la misma temperatura; a través de ellas no hay intercambio térmico (la diferencia de temperatura es 0). Las paredes en rojo (exteriores) son las que contribuyen a las pérdidas; a través del techo y el suelo se producirán otras pérdidas.
El cálculo se realiza considerando un ambiente a la vez. En el gráfico inferior se muestran tres ambientes, para cada uno de los cuales las pérdidas tendrán lugar a través de

— paredes
— ventanas y puertas exteriores
— suelo y techo
— renovaciones de aire

PLANO DE LOS TRES AMBIENTES CONSIDERADOS EN EL EJEMPLO



Por ejemplo, en el caso del ambiente 1, la situación es la siguiente:

- — paredes: Q2 + Q1 (Q2 en el lado AB y Q1 en el MA)
- — ventanas: una
- — suelo y techo: pérdidas proporcionales a la superficie del ambiente (lado AB × lado MA)

Para las paredes hay que tener en cuenta también la exposición solar. Si están orientadas hacia el norte (en el hemisferio norte) la cantidad de calor perdida ha de aumentarse en un 20%, hacia el este en un 15%, hacia el oeste en un 10% y ningún aumento para la orientación sur.
En el caso del ambiente 1, la pared AB está orientada hacia el este, por lo que la cantidad de calor calculada ha de ser aumentada en un 15%. La cantidad de calor se calcula con la fórmula (1).
Por ejemplo, si el lado AB mide 5 metros, AM = 4 metros y las paredes son de ladrillos de 38 cm (K = 1.3, ver tabla de la página 498), con una altura de 3.5 metros, tenemos:

$$\text{Area} = 5 \times 3.5 = 17.5 \text{ m}^2$$

de donde hay que restar la superficie de la ventana (2 m²), pues su contribución se evalúa por separado. Así, la superficie dispersora es:

$$\text{Area} = 17.5 - 2 = 15.5 \text{ m}^2$$

de donde, aplicando la fórmula (1):

Calor perdido (Q1) = 15.5 × 1.3 × 23
(23 = 18 + 5 = salto de temperatura)

Realizando los cálculos se obtiene:

Q1 = 463 kilocalorías/hora

Este valor ha de ser aumentado en un 15% (orientación este) y se convierte en:

Q1 = 533 kilocalorías/hora

A través de la ventana (superficie = 2 m², K = 5) se pierden: 2 × 5 × 23 = 230 kilocalorías/hora, que con el aumento del 15% se convierten en 265 kilocalorías/hora.
Por tanto, en total el calor perdido por la pared AB es 533 + 265 = 798 kilocalorías/hora.

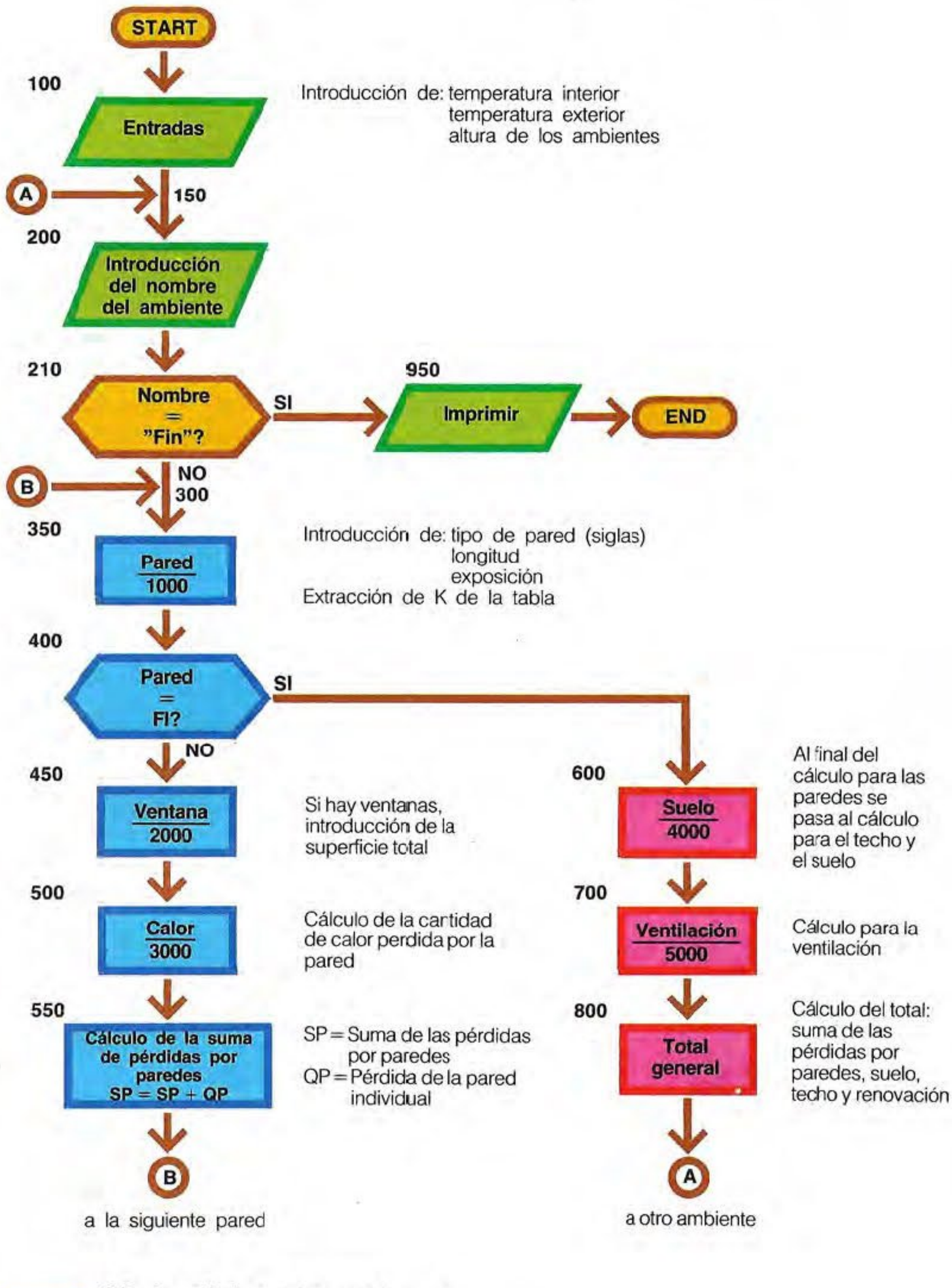
Para la pared MA se puede realizar un cálculo análogo.
También con la fórmula (1) se calcula la pérdida a través del techo (K = 2, área = lado AB × × lado MA) y del suelo (K = 1, misma área).
La suma de los valores calculados (paredes AB y MA, suelo y techo) da el calor total perdido a través de la estructura.
A esto hay que añadir el calor perdido por renovación del aire, debido al aire frío que entra.
Este calor (de ventilación) lo da la fórmula:

$$\text{Ventilación} \quad \left(\begin{array}{c}\text{Volumen}\\ \text{ambiente}\end{array}\right) \times 0.3 \times \left(\begin{array}{c}\text{Salto de}\\ \text{temperatura}\end{array}\right)$$

donde el volumen del ambiente es igual a la superficie del suelo multiplicada por la altura.
La suma del valor calculado anteriormente y el perdido en la ventilación da la cantidad de calor necesaria para mantener constante la temperatura en el ambiente considerado.
Las sumas de las cantidades de calor análogas calculadas para todos los ambientes da la potencia total de la instalación en kilocalorías/hora.
Con esto hemos finalizado la fase de análisis general. Ahora tenemos que sintetizar los distintos pasos y obtener el diagrama general.
El procedimiento de cálculo, aunque laborioso, se reduce a estos puntos fundamentales:

1 / Introducción de los valores de la temperatura externa y la interna, y de la altura de los ambientes
2 / Introducción del nombre del ambiente que se está considerando (si es FIN, el programa termina)
3 / Introducción del tipo de pared, longitud y orientación
4 / Si hay ventanas o puertas, introducción de su superficie
5 / Cálculo de la cantidad de calor perdida por la pared
6 / Si en el ambiente considerado hay más paredes se vuelve al punto 3
7 / Introducción de las superficies del suelo y del techo
8 / Cálculo de la cantidad de calor perdida a través del suelo y del techo
9 / Cálculo de la cantidad de calor perdida por la ventilación
10 / Total ambiente (paredes, techo, suelo y ventilación)
11 / Si hay otros ambientes se vuelve al punto 2
12 / Cálculo e impresión del total general

PROGRAMA DE CALCULO DE PERDIDAS TERMICAS: DIAGRAMA DEL MAIN



Cálculos relativos a las paredes
Cálculos relativos a un ambiente
Introducción de datos y presentación de resultados

La traducción en símbolos de las operaciones expuestas es el diagrama principal, y la codificación en Basic de las indicaciones contenidas en él suministra el main de este programa aplicativo. Los detalles de cada operación (por ejemplo, los puntos 5, 8, 9, etc.) se desarrollan mediante otras tantas subrutinas. El diagrama del main se muestra en la pág. 500; las subrutinas se indican mediante el símbolo de acción genérica (rectángulo), y para cada una de ellas se da una breve descripción de las funciones realizadas.

Las subrutinas

Para proseguir con el análisis y obtener el diagrama de cada rutina, antes hay que definir con exactitud qué magnitudes hay que suministrar a cada una de ellas, qué cálculos han de realizar y qué resultados han de proporcionar. Se ilustran ahora con detalle los distintos puntos.

Subrutina 1000: PARED
Funciones realizadas:

Lee las características de la pared
Controla si hay el tipo introducido por el operador (siglas de dos letras, tabla de pág. 498)
Toma de la tabla el valor de K
Predispone el factor multiplicativo correspondiente a la orientación

Salidas:

K = Coeficiente de intercambio térmico
P = Factor multiplicativo (P = 1.2 para el norte; 1.5 para el este; 1.1 para el oeste; 1 para el sur)
LP = Longitud de la pared

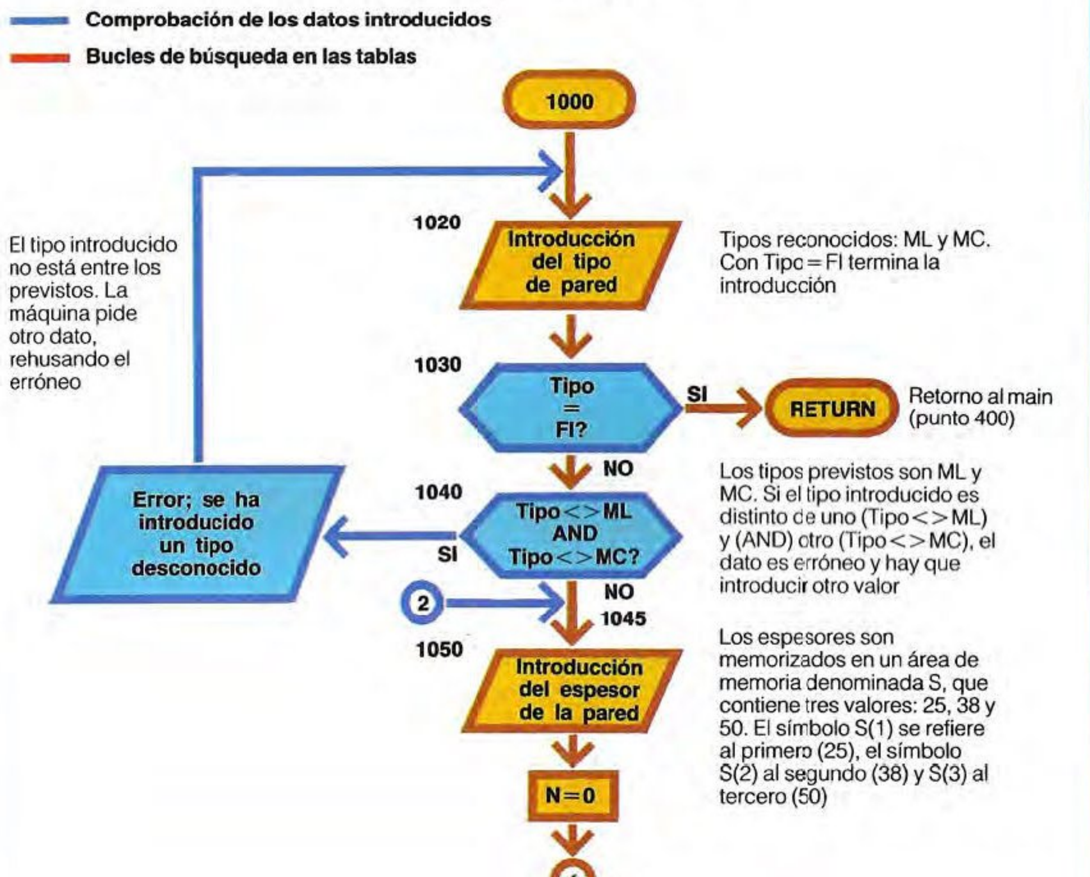
Tablas utilizadas:

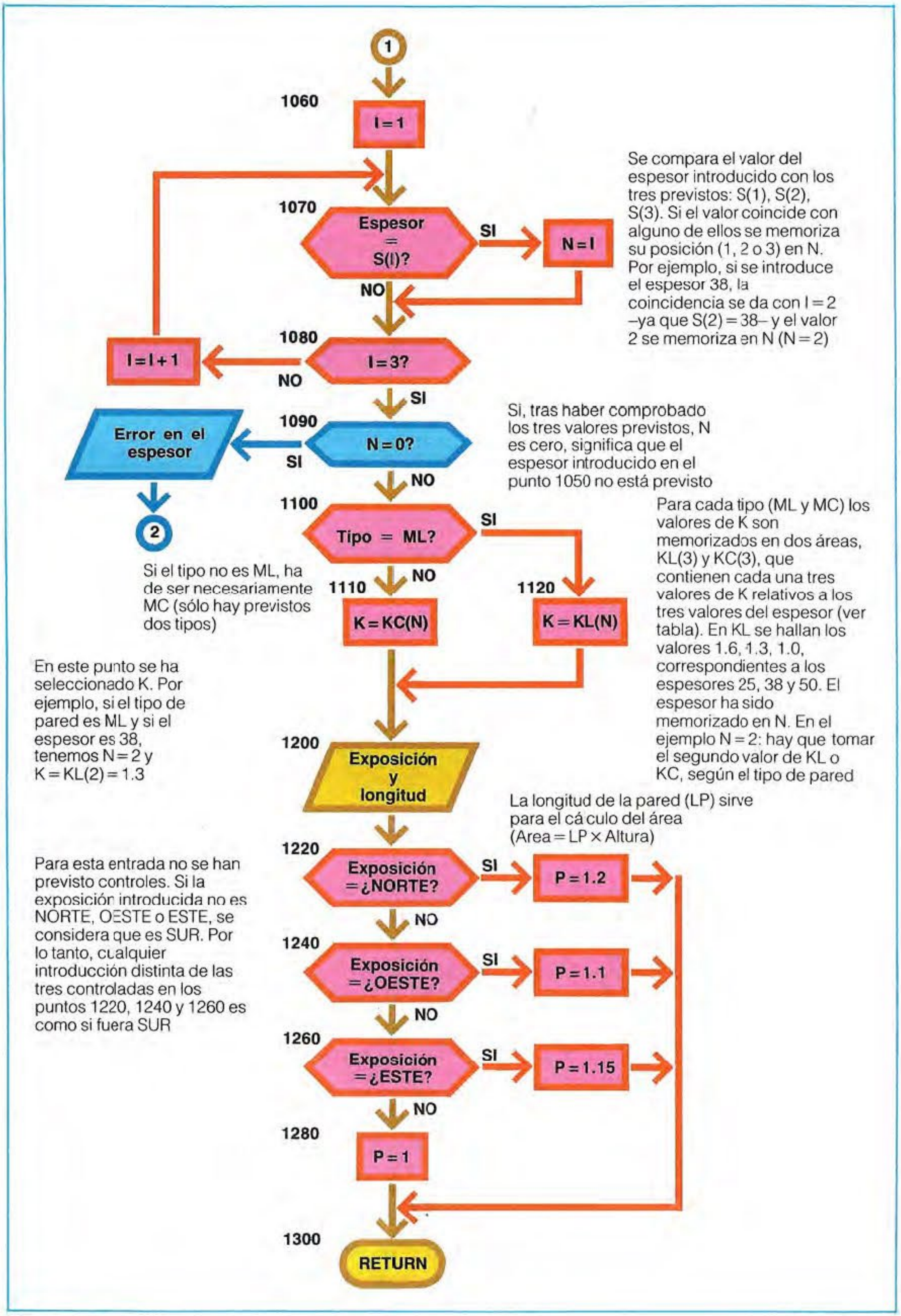
Parte de la tabla 1 (la relativa a las paredes)

Subrutina 2000: VENTANA
Funciones realizadas:

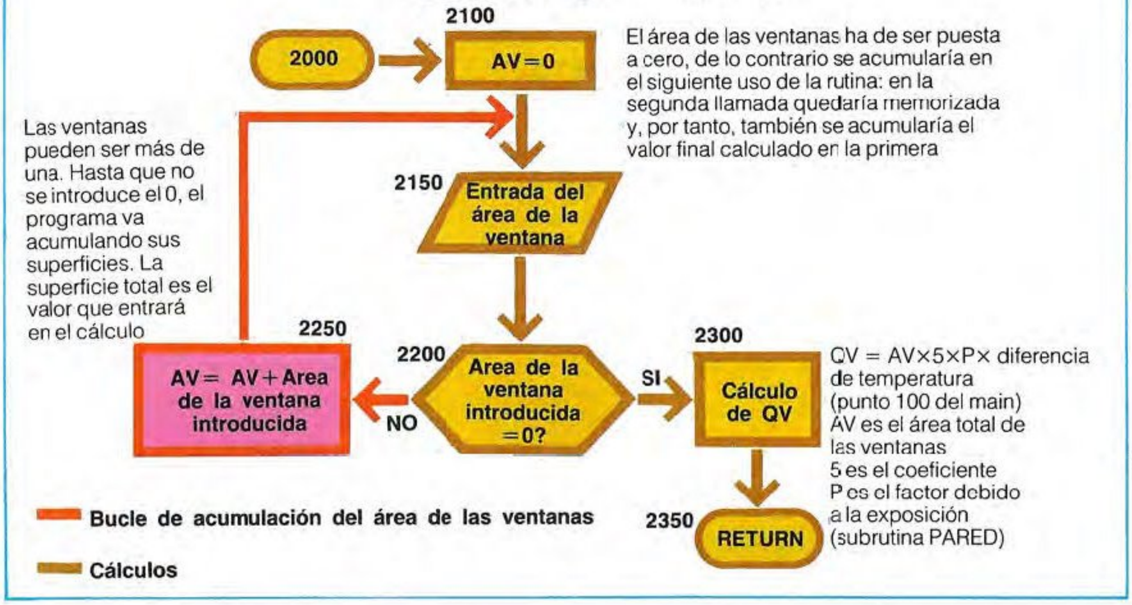
Introducción de la superficie total de eventuales ventanas

SUBRUTINA 1000: PARED




1
1060
I = 1
1070
Espesor = S(I)?
SI
N = I
NO
Se compara el valor del espesor introducido con los tres previstos: S(1), S(2), S(3). Si el valor coincide con alguno de ellos se memoriza su posición (1, 2 o 3) en N. Por ejemplo, si se introduce el espesor 38, la coincidencia se da con I = 2 –ya que S(2) = 38– y el valor 2 se memoriza en N (N = 2)
1080
I = 3?
NO
I = I + 1
SI
1090
N = 0?
SI
Error en el espesor
2
Si, tras haber comprobado los tres valores previstos, N es cero, significa que el espesor introducido en el punto 1050 no está previsto
NO
1100
Tipo = ML?
SI
NO
Si el tipo no es ML, ha de ser necesariamente MC (sólo hay previstos dos tipos)
1110
K = KC(N)
1120
K = KL(N)
Para cada tipo (ML y MC) los valores de K son memorizados en dos áreas, KL(3) y KC(3), que contienen cada una tres valores de K relativos a los tres valores del espesor (ver tabla). En KL se hallan los valores 1.6, 1.3, 1.0, correspondientes a los espesores 25, 38 y 50. El espesor ha sido memorizado en N. En el ejemplo N = 2: hay que tomar el segundo valor de KL o KC, según el tipo de pared
En este punto se ha seleccionado K. Por ejemplo, si el tipo de pared es ML y si el espesor es 38, tenemos N = 2 y K = KL(2) = 1.3
1200
Exposición y longitud
La longitud de la pared (LP) sirve para el cálculo del área (Area = LP × Altura)
1220
Exposición = ¿NORTE?
SI
P = 1.2
NO
Para esta entrada no se han previsto controles. Si la exposición introducida no es NORTE, OESTE o ESTE, se considera que es SUR. Por lo tanto, cualquier introducción distinta de las tres controladas en los puntos 1220, 1240 y 1260 es como si fuera SUR
1240
Exposición = ¿OESTE?
SI
P = 1.1
NO
1260
Exposición = ¿ESTE?
SI
P = 1.15
NO
1280
P = 1
1300
RETURN

SUBRUTINA 2000: VENTANA



Cálculo de la cantidad de calor perdido a través de las ventanas

Salidas:

AV = Area de las ventanas (a restar de la de la pared)

QV = Calor perdido a través de las ventanas

Subrutina 3000: CALOR

Funciones realizadas:

Cálculo del calor perdido por la pared y del total (pared más eventuales ventanas)

Salidas:

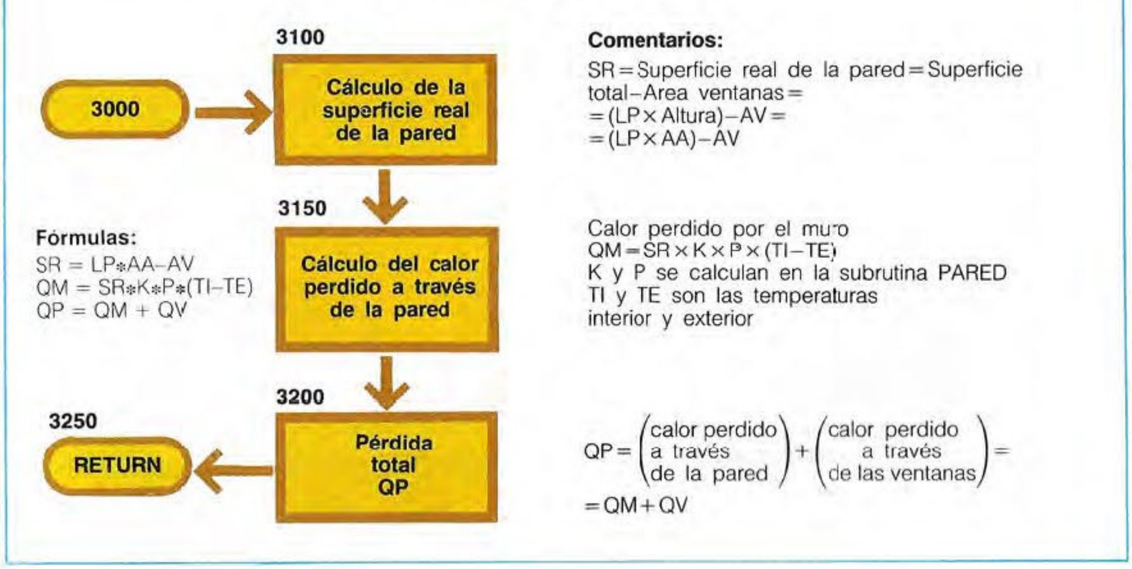
QP = Calor total perdido por la pared (muro + ventanas)

Subrutina 4000: SUELO

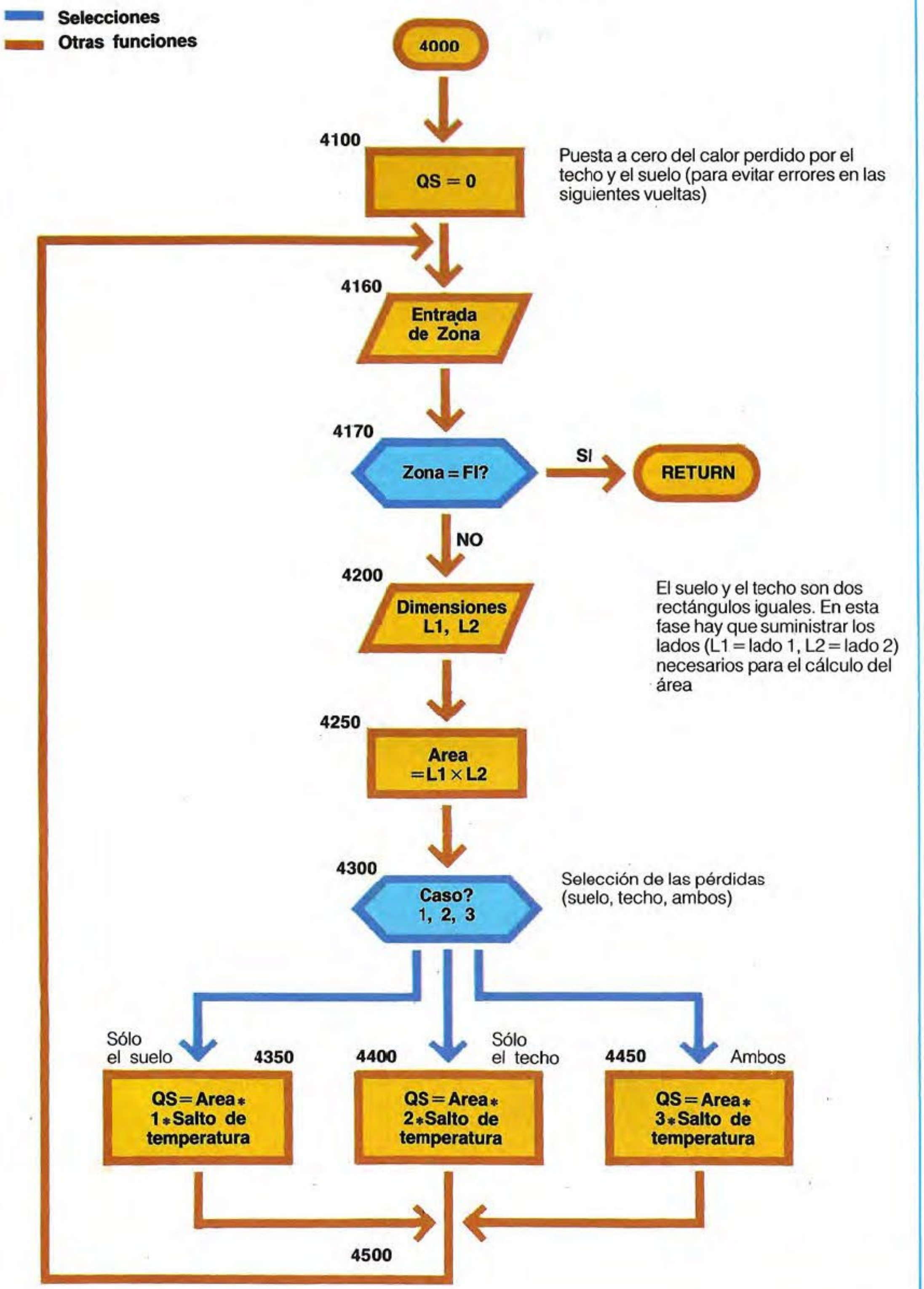
Funciones realizadas:

Cálculo de las pérdidas a través del suelo y el techo

SUBRUTINA 3000: CALOR



SUBRUTINA 4000: SUELO



Salidas:

QS = Pérdida total (suelo + techo).

En esta rutina debe pedirse si el suelo y el techo tienen pérdidas. Si los vanos dan a otros locales calentados de los pisos superior e inferior, hay que descontar las correspondientes pérdidas.

Subrutina 5000: RENOVACIONES

Funciones desarrolladas:

Cálculo del calor perdido en las renovaciones de aire.

Salidas

QR = Calor perdido en las renovaciones.

El programa

El listado completo del programa para el cálculo de las dispersiones térmicas se ha indicado en las páginas 505, 506, 507, 508 y 509, y está provisto de las necesarias explicaciones (insertadas como comentarios en las diversas líneas donde se ha visto la necesidad).

PROGRAMA DE CALCULO DE LAS DISIPACIONES TERMICAS

```
10 ' ** PROGRAMA DE CALCULO DISIPACIONES TERMICAS **
11 '
12 '
13 '
20 ' FILE = DISTER
21 '
22 '
23 '
28 ' ** ASIGNACIONES **
29 ' OPTION BASE 1
30 BI$=CHR$(27)+"+"+CHR$ (7)
32 '       TI  = TEMPERATURA INTERNA
34 '       TE  = TEMPERATURA EXTERNA
36 '       AA  = ALTURA AMBIENTES
38 '       NA$ = NOMBRE AMBIENTES
40 '       S   = AREA DE MEMORIA DEDICADA A ESPESORES
42 '       KM  = COEFICIENTE TERMICO MAMPOSTERIA
44 '       KR  = COEFICIENTE TERMICO REVOQUE
46 '       TP$ = TIPO DE LA PARED (ML/MC)
48 '       SP  = ESPESOR PARED
50 '       ES$ = TIPO DE EXPOSICION (N.-S.-E.-O.)
52 '       LP  = LONGITUD PARED
54 '       P   = FACTOR MULTIPLICADOR QUE DEPENDE DEL ESPESOR
56 '       AI  = AREA DE VENTANA SENCILLA
58 '       AV  = AREA TOTAL VENTANAS
60 '       QV  = CALOR DISPERSADO A TRAVES DE LAS VENTANAS
62 '       SR  = SUPERFICIE REAL DE LA PARED
64 '       QM  = CALOR DISPERSADO A TRAVES DEL MURO
66 '       QP  = SUMA DE QM+QV (MURO+VENTANAS)
68 '       ST  = SUMATORIO DE TODOS LOS (QP)
70 '       NU$ = NUMERO DE ZONA
72 '       L1  = LONGITUD 1 LADO SUELO
74 '       L2  = LONGITUD 2 LADO SUELO
76 '       A1  = AREA L1*L2
78 '       AT  = SUMA DE LAS AREAS (DE LAS VARIAS ZONAS DE SUELO)
80 '       QS  = CALOR DISPERSADO A TRAVES DE LOS SUELOS Y TECHOS
82 '       QR  = CALOR DISPERSADO EN LAS RENOVACIONES DE AIRE
84 '       T1  = SUMATORIO DE TODOS LOS CALORES DISPERSADOS = POTENCIA
86 '             DE LA INSTALACION TERMICA
88 '       T2  = SUMATORIO DE TODOS LOS (QR)
90 '       T3  = SUMATORIO DE TODOS LOS (QS)
92 '       T4  = SUMATORIO DE TODOS LOS (QP)
100 INPUT "TEMPERATURA INTERNA";TI
110 INPUT "TEMPERATURA EXTERNA";TE
120 INPUT "ALTURA DE LOS AMBIENTES";AA
150 PRINT BI$
152 PRINT "INTRODUCIR LA PALABRA (FIN) PARA TERMINAR"
200 INPUT "NOMBRE DEL AMBIENTE";NA$
210 IF NA$="FIN" THEN GOTO 950
300 '
350 GOSUB 1000                               'RUTINA PARED
400 IF TP$="FI" THEN GOTO 600
450 GOSUB 2000                               'RUTINA VENTANA
500 GOSUB 3000                               'RUTINA CALOR
550 ST=ST+QP
```

```
560 GOTO 300
600 GOSUB 4000                              'RUTINA SUELO
700 GOSUB 5000                              'RUTINA RENOVACION AIRE
800 PT=ST+QS+QR
802 LPRINT "TOTAL CALOR DISPERSADO POR EL LOCAL";NA$;":";PT:ST=0
804 LPRINT CHR$ (12)    'INSTRUCCION EN CODIGO QUE PERMITE A LA IMPRESORA
805 '                    IR AL PRINCÍPIO DE LA PÁGINA SIGUIENTE
806 T1=T1+PT      'T1=MEMORIA DE ACUMULACION PARA EL CALCULO DE LA POTENCIA'
807 '                DE LA INSTALACION TERMICA

808 PT=0          'PUESTA A CERO DE (PT) PARA EVITAR ERRORES EN GIROS SUCESIVOS
900 GOTO 150
950 ' ** INSTRUCCIONES FINAL DE IMPRESÍON **
955 LPRINT "TEMPERATURA INTERNA = ";TI
960 LPRINT "TEMPERATURA EXTERNA = ";TE
965 LPRINT "ALTURA DE LOS AMBIENTES = ";AA
970 LPRINT
972 LPRINT "CALOR TOTAL DISPERSADO (calorías/hora):"
974 LPRINT
975 LPRINT "PAREDES Y VENTANAS= ";T4:LPRINT
976 LPRINT "SUELOS Y TECHOS= ";T3
978 LPRINT
980 LPRINT "RENOVACIONES DE AIRE= ";T2
982 LPRINT
984 LPRINT "POTENCIA DE LA INSTALACION TERMICA= ";T1
986 END
1000 ' ** SUBRUTINA PARED **
1001 S(1)=25
1002 S(2)=38
1003 S(3)=50
1004 KM(1)=1.6
1005 KM(2)=1.3
1006 KM(3)=1
1007 KC(1)=2.2
1008 KC(2)=1.8
1009 KC(3)=1.6
1010 PRINT BI$
1012 PRINT "SUBRUTINA PARED    CALCULO DISPERSION DE CALOR"
1014 PRINT
1016 PRINT NA$
1017 PRINT:PRINT "SI NO HAY OTROS LADOS DISPERSANTES: INTRODUCIR RETURN"
1018 INPUT "LADO";L$    'L$ MEMORIZA EL NOMBRE DEL LADO (PARED)
1020 INPUT "TIPO PARED (ML/MC/VE)";TP$
1030 IF TP$="FI" THEN RETURN
1040 IF TP$<> "ML" AND TP$<> "MC" THEN PRINT "ERROR EN TIPO PARED":GOTO 1020
1045 '
1050 INPUT "ESPESOR DE LA PARED" (25/38/50)";SP
1055 N=0
1060 I=1
1070 IF SP=S(I) THEN N=I
1080 IF I=3 THEN GOTO 1090 ELSE I=I+1:GOTO 1070
1090 IF N=0 THEN PRINT "ERROR EN EL ESPESOR":GOTO 1045
1100 IF TP$="ML" THEN GOTO 1120
1110 K=KC(N):GOTO 1200
1120 K=KM(N)
1200 INPUT "EXPOSICION (NORTE/SUR/ESTE/OESTE)";ES$
1210 INPUT "LONGITUD PARED en Mt.";LP
1220 IF ES$="NORTE" THEN P=1.2:GOTO 1282
1240 IF ES$="OESTE" THEN P=1.1:GOTO 1282
1260 IF ES$="ESTE" THEN P=1.15:GOTO 1282
1280 P=1
1282 LPRINT:LPRINT "AMBIENTE : ";NA$
1283 LPRINT "PARED LADO" : ";L$
1284 LPRINT "K = ";K
1288 LPRINT "FACTOR MULTIPLICADOR PARA LA EXPOSICION P = ";P
1292 LPRINT "LONGITUD PARED Mt. =;LP
1300 RETURN
2000 ' ** SUBRUTINA VENTANA **
2010 PRINT BI$
2020 PRINT "RUTINA VENTANA          CALCULO DISPERSION DE CALOR"
2030 PRINT
2040 PRINT NA$
2100 AV=0
```

```
2150 INPUT "AREA DE LA VENTANA en Mc. (INTRODUCIR 0 PARA TERMINAR)";AI
2200 IF AI=0 THEN GOTO 2300
2250 AV=AV+AI
2260 GOTO 2150
2300 QV=AV*5*P*(TI-TE)     'PARA LAS VENTANAS K=5      P=VER ASIGNACIONES
2306 LPRINT "AREA TOTAL VENTANAS Mc. = ";AV
2310 LPRINT "CALOR DISPERSADO POR LAS VENTANAS calorías/hora = ";QV
2350 RETURN
3000 ' ** SUBRUTINA CALOR **
3100 SR=(LP*AA)-AV
3150 QM=SR*K*P*(TI-TE)
3200 QP=QM+QV
3201 LPRINT "CALOR DISPERSADO POR EL MURO calorías/hora = ";QM
3202 LPRINT "PERDIDA DE CALOR (MURO+VENTANAS) calorías/hora = ";QP
3204 T4=T4+QP
3250 RETURN
4000 ' ** SUBRUTINA SUELO **
4010 PRINT BI$            'BI$=BORRA EL VIDEO Y EMITE UN SONIDO
4020 PRINT "RUTINA SUELO              CALCULO DISPERSION DE CALOR"
4030 PRINT                'ESPACIADO EN EL VIDEO
4040 PRINT NA$            'INTRODUCIR EN EL VIDEO NOMBRE DEL AMBIENTE A EXAMEN
4100 QS=0                 'PUESTA A CERO INICIAL PARA EVITAR ERRORES
4101 '                     EN GIROS SIGUIENTES
4110 AT=0
4112 PRINT "NOTA: EN CUALQUIER CASO SE INTRODUCE EL NUMERO DE ZONA EN QUE"
4113 PRINT "SE HA SUBDIVIDIDO EL SUELO"
4160 INPUT "NUMERO DE ZONA (INTRODUCIR FI PARA TERMINAR);+NU$
4170 IF NU$="FI" THEN RETURN
4200 INPUT "PRIMER LADO DEL SUELO";L1
4210 INPUT "SEGUNDO LADO DEL SUELO";L2
4250 A1=L1*L2             'A1=AREA DE LA ZONA SENCILLA
4260 AT=AT+A1             'AT=AREA TOTAL DEL SUELO
4261 PRINT "INTRODUCIR UNO DE LOS SIGUIENTES NUMEROS:"
4262 PRINT "0 = SUELO Y TECHO NO DISIPAN"
4263 PRINT "1 = SOLO DISIPA EL SUELO"
4264 PRINT "2 = SOLO DISIPA EL TECHO"
4265 PRINT "3 = DISIPAN LOS DOS"
4266 INPUT "INTRODUCIR OPCION ELEGIDA",C
4267 IF C=0 THEN RETURN
4268 ' *** LA INSTRUCCION: ON C GOTO 4350, 4400, 4450 ES EQUIVALENTE A LAS TRES
4270 '      INSTRUCCIONES QUE SIGUEN (4300-4310-4320) ***
4300 IF C=1 THEN GOTO 4350     'SOLO DISIPA EL SUELO
4310 IF C=2 THEN GOTO 4400     'SOLO DISIPA EL TECHO
4320 IF C=3 THEN GOTO 4450     'DISIPAN LOS DOS
4350 QS=AT*1*(TI-TE)           'QS=CALOR DISPERSADO A TRAVES TECHO Y SUELO
4351 LPRINT:LPRINT "OTRAS DISPERSIONES"
4352 LPRINT "CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = ";QS
4354 T3=T3+QS
4360 GOTO 4160
4400 QS=AT*2*(TI-TE)
4401 LPRINT:LPRINT "OTRAS DISPERSIONES"
4402 LPRINT "CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = ";QS
4404 T3=T3+QS
4410 GOTO 4160
4450 QS=AT*3*(TI-TE)
4451 LPRINT:LPRINT "OTRAS DISPERSIONES"
4452 LPRINT "CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = ";QS
4454 T3=T3+QS
4460 GOTO 4160
4500 RETURN
5000 ' ** SUBRUTINA RENOVACION **
5002 ' Calor perdido durante las renovaciones de aire =
5004 ' Volumen Ambiente * 0.3 * Salto de Temperatura (TI-TE)
5010 QR=AT*AA*,3*(TI-TE)           'QR = CALOR PERDIDO DURANTE RENOVACIONES AIRE
5016 LPRINT "CALOR DISPERSADO DURANTE RENOVACIONES AIRE calorías/hora = ";QR

5018 T2=T2+QR
5020 RETURN
```

```
AMBIENTE : CUARTO DE ESTAR-COMEDOR
PARED LADO : BC
K = 1
FACTOR MULTIPLICADOR PARA LA EXPOSICION P = 1.15
LONGITUD PARED Mt. = 3.5
AREA TOTAL VENTANAS Mc. = 1.8
CALOR DISPERSADO POR LAS VENTANAS calorías/hora = 134.55
CALOR DISPERSADO POR EL MURO calorías/hora = 130.065
PERDIDA DE CALOR (MURO+VENTANAS) calorías/hora = 264.615

AMBIENTE: CUARTO DE ESTAR-COMEDOR
PARED LADO : CD
K = 1
FACTOR MULTIPLICADOR PARA LA EXPOSICION P = 1
LONGITUD PARED Mt. = 4
AREA TOTAL VENTANAS Mc. = 2.52
CALOR DISPERSADO POR LAS VENTANAS calorías/hora = 163.8
CALOR DISPERSADO POR EL MURO calorías/hora 123.24
PERDIDA DE CALOR (MURO+VENTANA) calorías/hora = 287.04

OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = 136.5

OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = 396.5
CALOR DISPERSADO DURANTE LAS RENOVACIONES DE AIRE calorías/hora = 178.425
TOTAL CALOR DISPERSADO EN LOCAL CUARTO ESTAR-COMEDOR : 1126.58


AMBIENTE : BAÑO
PARED LADO : DE
K = 1
FACTOR MULTIPLICADOR PARA LA EXPOSICION P = 1
LONGITUD PARED Mt. = 1.8
AREA TOTAL VENTANA Mc. = 1.2
CALOR DISPERSADO POR LAS VENTANAS calorías/hora = 78
CALOR DISPERSADO POR EL MURO calorías/hora = 54.6
PERDIDA DE CALOR (MURO+VENTANAS) calorías/hora = 132.6

OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = 117
CALOR DISPERSADO EN LAS RENOVACIONES DE AIRE calorías/hora = 52.65
TOTAL CALOR DISPERSADO POR EL LOCAL BAÑO : 302.25


AMBIENTE : COCINA
PARED LADO : EF
K = 1
FACTOR MULTIPLICADOR PARA LA EXPOSICION P = 1
LONTITUD PARED Mt. = 2.6
AREA TOTAL VENTANAS Mc. = 2.52
CALOR DISPERSADO POR LAS VENTANAS calorías/hora = 163.8
CALOR DISPERSADO POR EL MURO calorías/hora = 68.64
PERDIDA DE CALOR (MURO + VENTANAS) calorías/hora = 232.44

OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES DEL TECHO Y SUELO calorías/hora = 169
CALOR DISPERSADO EN LAS RENOVACIONES DE AIRE calorías/hora = 76.05
TOTAL CALOR DISPERSADO EN EL LOCAL COCINA : 477.49


AMBIENTE : DORMITORIO
PARED LADO : FG
K = 1
FACTOR MULTIPLICADOR PARA LA EXPOSICION P = 1
LONGITUD PARED Mt. = 3.5
AREA TOTAL VENTANAS Mc. = 1.8
CALOR DISPERSADO POR LAS VENTANAS calorías/hora = 117
CALOR DISPERSADO POR EL MURO calorías/hora =113.1
PERDIDA DE CALOR (MURO + VENTANAS) calorías/hora = 230.1

OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES TECHO Y SUELO calorías/hora = 156
```

```
OTRAS DISPERSIONES
CALOR DISPERSADO A TRAVES DE TECHO Y SUELO calorias/hora = 338
CALOR DISPERSADO DURANTE LAS RENOVACIONES DE AIRE calorías/hora = 152.1
TOTAL CALOR DISPERSADO EN EL LOCAL DORMITORIO = 720.2

TEMPERATURA INTERNA = 18
TEMPERATURA EXTERNA = 5
ALTURA DE LOS AMBIENTES = 3

CALOR TOTAL DISPERSADO (calorías/hora):

MUROS Y VENTANAS = 1146.8

SUELOS Y TECHOS = 1313

RENOVACIONES DE AIRE = 459.225

POTENCIA DE LA INSTALACION TERMICA = 2626.52
```

En la formulación de los mensajes de salida se ha adoptado «caloría» para indicar la «caloría grande» (1 caloría grande = 1000 calorías pequeñas = 1 kilocaloría).
En la secuencia fotográfica (págs. 510 a 514) se ha ilustrado de forma limitada la fase de ejecución correspondiente a la entrada de datos. Aquí se refiere constantemente al cálculo de las dispersiones térmicas del apartamento del ejemplo (ver planta en pág. 515).

Los colores de esta fotografía demuestran insuficiencia o falta total de aislamiento térmico.

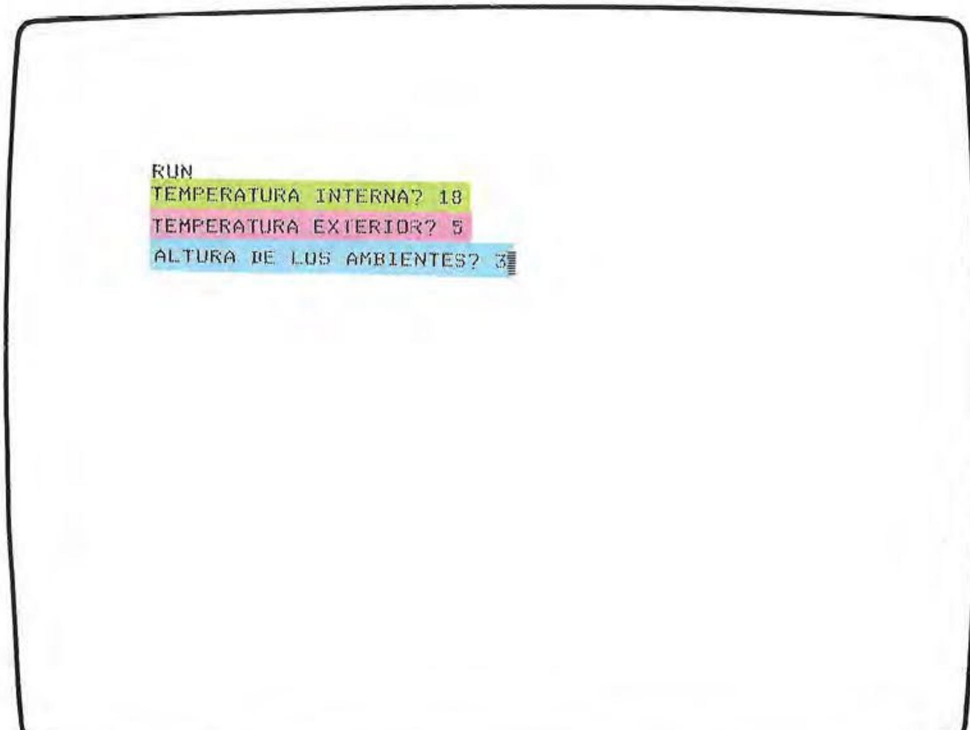


AGA Infrared Systems/Science Photo Library

PROGRAMA PARA EL CALCULO DE LAS DISIPACIONES TERMICAS FASE DE INTRODUCCION DE DATOS (1)



La ejecución del programa es activada por la introducción del comando RUN. El intérprete procede entonces a la traducción y a la ejecución de las líneas de programa en sucesión, partiendo de la que tiene el label más pequeño.



Cuando el control pasa a la línea 100, la solicitud de input de teclado es interpretada y realizada (la función INPUT está ilustrada de forma más amplia en el capítulo dedicado a las funciones I/O). La ejecución del programa se detiene y el sistema se pone en espera del dato requerido.
En la pantalla aparece el mensaje que ha insertado el programador en la línea 100, seguido de un interrogante (colocado por el sistema). El operador digita en el teclado el valor 18 y pulsa la tecla RETURN: el valor introducido es asignado a la variable TI, como especifica la línea 100.

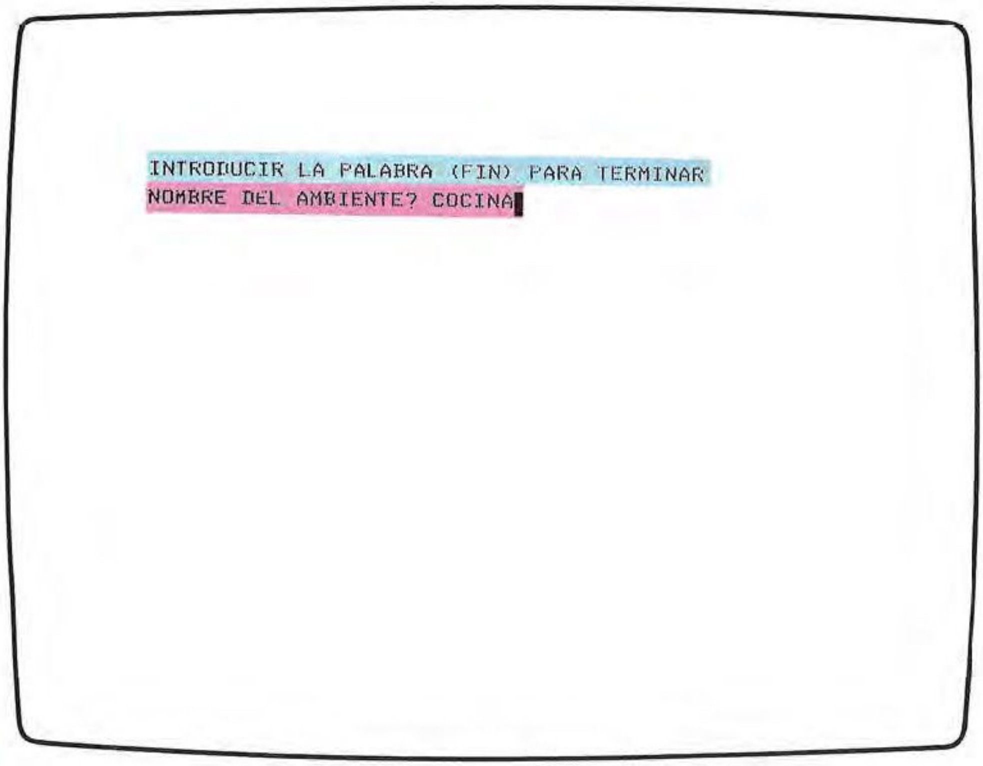
El programa pide la introducción de un nuevo dato (línea 110) y el operador digita el valor 5, que será asignado a la variable TE.

A la siguiente petición de introducción (línea 120), el operador proporciona la altura de los ambientes (3 metros). En el momento en que se tomó esta fotografía aún no se había pulsado la tecla RETURN. Por esta razón, el cursor todavía está posicionado al final de la última línea.

PROGRAMA PARA EL CALCULO DE LAS DISIPACIONES TERMICAS FASE DE INTRODUCCION DE DATOS (2)

En esta fase se efectúa la selección del ambiente a considerar en los cálculos de dispersión.





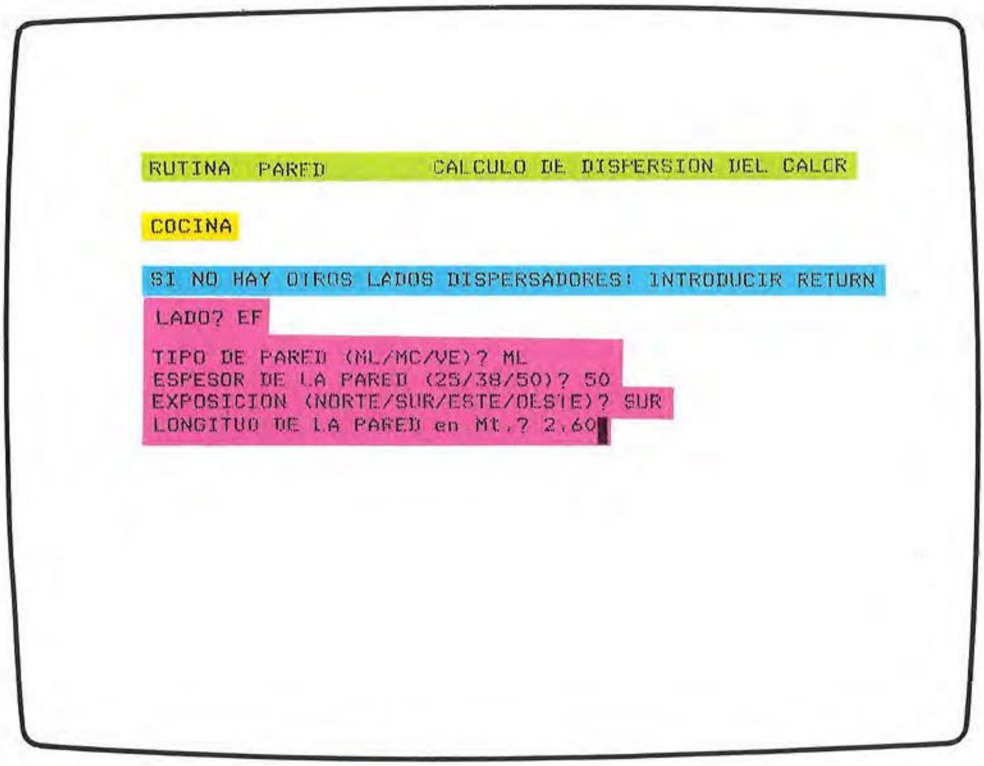
Este mensaje se imprime a continuación de la ejecución de la línea 152. El programador ha previsto la posibilidad de terminar la ejecución introduciendo por la consola el nombre ficticio FIN y advierte al operador con el mensaje presentado. Debe observarse que el único modo para terminar de forma «normal» la ejecución de aquél es seguir la indicación proporcionada.
La práctica adoptada en este ejemplo es muy corriente, y es típica de todos los programas de aplicación de tipo interactivo. El usuario del programa, en este caso, ha seguido al pie de la letra las indicaciones proporcionadas durante el RUN del mismo programa, con el fin de evitar que durante la ejecución se establezcan recorridos no previstos. Está claro que en fase de programación deben hacerse todos los esfuerzos necesarios para guiar la elección del usuario final, introduciendo todos los controles necesarios para evitar el establecimiento de condiciones anómalas.
Potencialmente, una gestión equivocada de la interactividad puede llevar a la pérdida de datos preciosos.

Después del mensaje de aviso, la línea 200 pide el input por consola del nombre del ambiente que será considerado en la siguiente elaboración. El operador introduce la cadena COCINA que está asignada a la variable NA$.

PROGRAMA PARA EL CALCULO DE LAS DISIPACIONES TERMICAS FASE DE INTRODUCCION DE DATOS (3)

Después de la introducción de los datos comunes a todos los ambientes [fase (1)] y del nombre del ambiente particular a considerar [fase (2)], por efecto de la línea 350 el control pasa a la subrutina 1000 (SUBRUTINA PARED).





Este mensaje es presentado en la pantalla debido a la línea 1012.

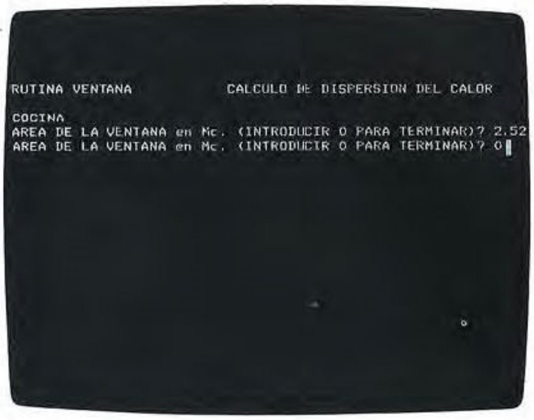
La línea 1016 produce la presentación en pantalla del nombre del ambiente considerado, que sirve principalmente como recordatorio al operador.

El siguiente mensaje proporciona las indicaciones para salir del bucle de petición de los datos correspondientes a cada pared sencilla dispersante del local considerado. La introducción del código RETURN producirá el paso del control a la fase de petición del nombre del nuevo ambiente a considerar.

En esta fase, el operador introduce uno tras otro los datos pedidos por el programa en las líneas 1018, 1020, 1050, 1200 y 1210. Puede observarse cómo la introducción de los datos está estrechamente «guiada» por los mensajes presentados en pantalla utilizando la función INPUT.

PROGRAMA PARA EL CALCULO DE LAS DISIPACIONES TERMICAS FASE DE INTRODUCCION DE DATOS (4)

Si en el transcurso de la ejecución de la subrutina 1000 se introduce la cadena VE en respuesta a la petición del tipo de pared, el control vuelve al main, que lo pasa a la subrutina VENTANA. El código VE señala de hecho que en el ambiente en curso de elaboración existe al menos una ventana, que debe calcularse aparte.





Esta inscripción es visualizada por efecto de la línea 2020.

El nombre del ambiente está repetido para comodidad del operador.

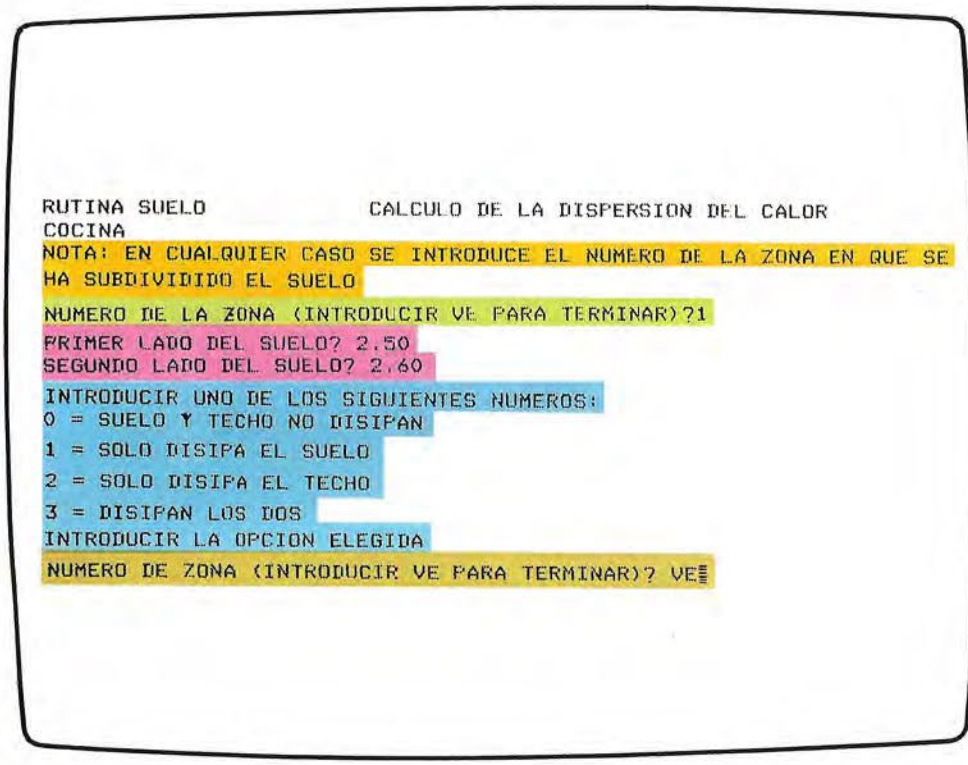
En esta línea se inicia la fase de cálculo de las superficies de las ventanas que hay en el ambiente. En este caso existe una sola ventana (ver el plano del apartamento).

La introducción del valor 0 por el operador determina el paso del control a la fase de cálculo del calor dispersado a través de las ventanas (línea 2300).

PROGRAMA PARA EL CALCULO DE LAS DISIPACIONÈS TERMICAS FASE DE INTRODUCCION DE DATOS (5)

La pantalla muestra la fase siguiente al paso del control a la subrutina SUELO, paso activado en la línea 600 del main.





Este mensaje se presenta en pantalla desde las líneas 4112 y 4113, y tiene el objeto de evitar el establecimiento de condiciones de error debidas a la falta de introducción de un dato necesario para el programa.
La subdivisión del suelo en zonas es necesaria debido a que la planta del ambiente no es rectangular (o cuadrada). En este caso conviene descomponerla en un conjunto de más zonas rectangulares (o cuadradas), para cada una de las cuales se deben introducir los datos necesarios para el cálculo de la superficie.

El mensaje pide que se introduzca el número de zona incluso cuando sólo existe una sola.

Introducción del número de zona.

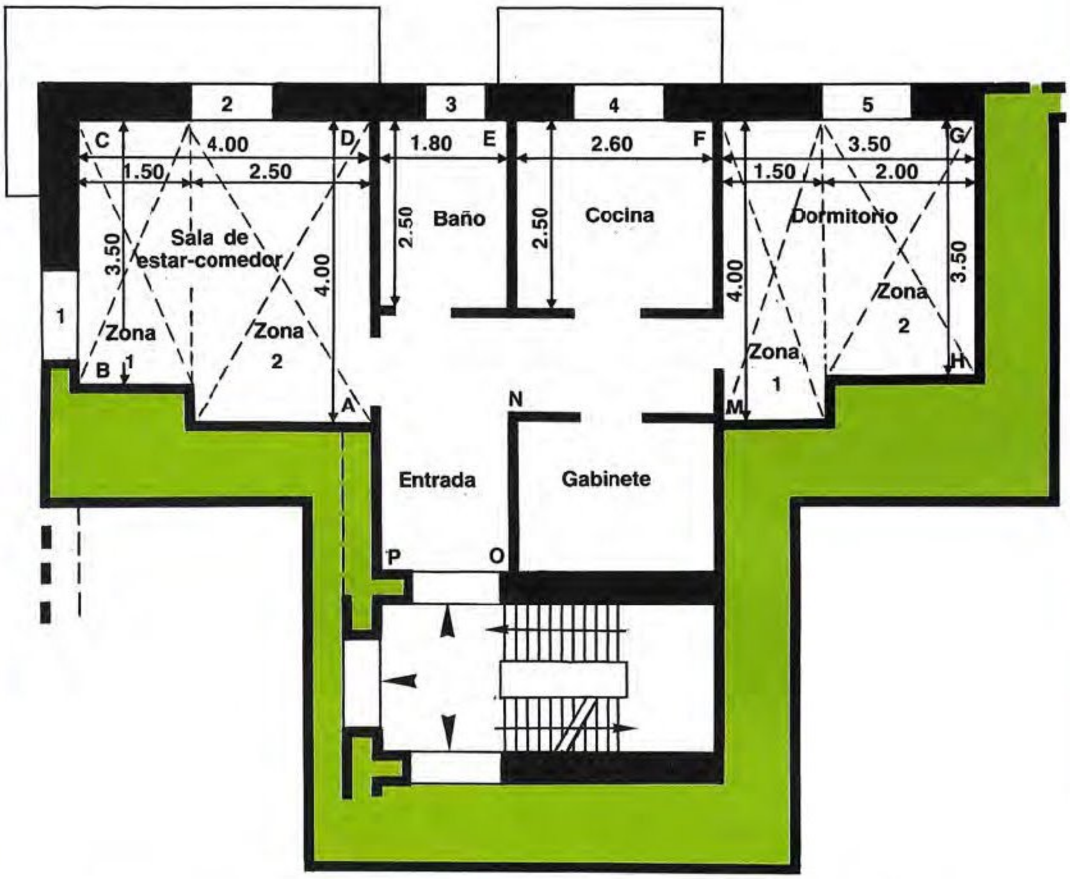
Introducción de las longitudes de los lados de cada zona.

En esta fase, el programa pide al operador que seleccione la situación en que se encuentra la zona del suelo examinada.

La respuesta de la consola indica que la dispersión térmica se produce sólo a través del techo.

Después del cálculo del calor dispersado en la zona del suelo examinado, la línea 4360 pasa el control a la 4160, y se repite la petición del número de la eventual segunda zona a considerar. La respuesta del operador determina el retorno al main.

PLANTA DEL APARTAMENTO



Nota: Las paredes que dispersan el calor son BC, CD, DE, EF y FG; las que dan a otros apartamentos no lo dispersan.

Leyenda: 1 = Ventana de 1.80 m²
2 = Puerta ventana de 2.52 m²
3 = Ventana de 1.20 m²
4 = Puerta ventana de 2.52 m²
5 = Ventana de 1.80 m²

Suelo del cuarto de estar-comedor:	Zona 1 = 1.50×3.50
	Zona 2 = 2.50×4.00
Suelo del dormitorio	Zona 1 = 1.50×4.00
	Zona 2 = 2.00×3.50
Suelo del baño	= 1.80×2.50
Suelo de la cocina	= 2.60×2.50

La vaca lechera computerizada

En el mundo viven cerca de 210 millones de vacas lecheras, cuya producción anual de leche se estima en 427 millones de toneladas, de las cuales 5 millones de toneladas son de leche en polvo; de mantequilla 7 millones de toneladas y de queso 11 millones de toneladas. El transporte y la distribución en el mercado de cantidades tan considerables de productos alimentarios plantean grandes problemas a los gobiernos de los diversos países y, en particular, a los de la Comunidad Económica Europea.
Para hacer frente a las variaciones de la política y a los crecientes costes de la elaboración de los comestibles y de su transporte, los criadores y agricultores deben programar la producción con una precisión que nunca se había visto antes. En consecuencia, junto a las vacas, en las granjas ha hecho su aparición el calculador electrónico. El ordenador contribuye a la gestión de criadores de vacas lecheras.
La previsión de los costes para los gastos generales, para las inversiones de capital, para la alimentación y para el mantenimiento es de extrema importancia para una eficiente gestión de las exportaciones. Con este fin, en los Países Bajos, Irlanda septentrional, en el Eire e incluso en el Reino Unido, se ha puesto a punto un intento de programación que se vale del procesador para prever el comportamiento de los animales para todo el transcurso del año. Los productores comunican a un centro de proceso, situado en Gran Bretaña, la producción mensual de leche de cada vaca lechera además de los datos correspondientes a determinados sucesos, como los partos y los apareamientos. A su vez, el procesador programa la producción de leche y las características de la alimentación de cada vaca lechera para el próximo mes y prevé la fecha probable en que cada vaca cesará de producir leche. En Estados Unidos se han puesto a punto programas análogos, en particular con el auxilio de los University Agricultural Extension Services Centres.
Uno de estos centros calcula las características de la alimentación invernal, transmitiendo telefónicamente las correspondientes informaciones a los operadores interesados, sirviéndose de una voz femenina simulada.
Otro sistema realizado en Estados Unidos, en el Agricultural Economic Departament de la Michigan State University, prevé el empleo del Telplan, sistema de subdivisión de tiempo accesible a través de un teléfono dotado de un teclado especial para la introducción de los datos en el procesador una vez que se ha establecido la comunicación.
El Telplan ofrece al operador interesado cerca de 50 programas individuales elaborados con el calculador, los más importantes de los cuales corresponden a la reducción al mínimo del coste de las raciones alimentarias, la preparación de balances preventivos y la presentación de propuestas de inversión de capital. Entre los demás programas realizables con el Telplan se incluyen la estimación de las características de raza y de los costes de producción por cada vaca lechera.
En Canadá, en el intento de coordinar la producción de adecuados programas de gestión con su aplicación, en el año 1965, el gobierno federal ha establecido el Canfarm. Pocos años después de su entrada en vigor, cerca de 10.000 productores se han adherido al Canfarm y, hoy, el conjunto de los programas de aplicación a disposición de los adheridos comprenden informaciones sobre las raciones de pienso, la planificación de la cabaña de ganado, precisiones sobre la circulación del dinero y modelos de proyecto de crianza. Más recientemente, también en Canadá, se ha introducido una combinación comercial para la selección de los apareamientos mediante terminales conducidos hasta las instalaciones de los productores desde el procesador central.
En el Reino Unido, por ejemplo, actualmente más de un 20% de los productores lecheros está asociado con alguna forma de gestión computerizada de la empresa. La mayor parte de los adheridos envía los datos por correo a un calculador central que los elabora. La unidad central tiene el tamaño de un gran armario y un coste de muchos millones.
Actualmente se va afirmando una metodología diferente de intervención, basada en el empleo de sistemas de gestión más pequeños, económicos y descentralizados. De hecho, el ordenador personal parece más de acuerdo con la naturaleza de las necesidades que deben ser satisfechas en una instalación de vacas lecheras. El desarrollo de adecuados paquetes de aplicación y el acoplamiento del ordenador con determinados aparatos de adquisición de datos permiten posibilidades fascinantes. Uno de los sistemas más avanzados para la gestión automati-

zada de la factoría, el AFMS/80 de Fullwood Ellesmere Electronics, ofrece un paquete de aplicación que utiliza la tecnología del microprocesador y que prevé la adquisición de una vasta gama de informaciones relativas a la gestión y a las prestaciones de los animales.

Para quedar conectada al sistema, cada vaca de la vacada lleva alrededor del cuello un dispositivo de identificación. Al entrar en la sección de ordeñado, o al salir, este dispositivo queda «marcado» mediante una bobina dispuesta adecuadamente –por ejemplo, a la entrada de los departamentos– de manera que se registra la identidad de la vaca por medio de una señal que pertenece únicamente a un animal determinado. La señal es recibida e interpretada, a través de una antena, por un receptor conectado con el microprocesador que realiza la identificación, permitiendo así la individualización automática en cualquier momento de la alimentación y del ordeñado de cada vaca, así como el registro de la cantidad de comida ingerida y de leche producida por cada cabeza de ganado. Como el sistema conoce el departamento asignado a cada animal y la necesidad alimentaria de cada vaca, esto proporciona automáticamente la cantidad óptima de comida aconsejable caso por caso. De manera análoga, cuando la vaca se acerca a uno de los comederos del exterior del establo, recibirá la cantidad justa de concentrado alimenticio que necesita. Más adelante, la vaca pasa por encima de una balanza automática que pesa al animal, registra su identidad y comunica los datos al procesador central.

El control de la alimentación produce beneficios tanto a la vaca como a quien trabaja en la facto-



G.R. Roberts



Alan Hutchison Library

Arriba. Instalación para la producción de leche en polvo.

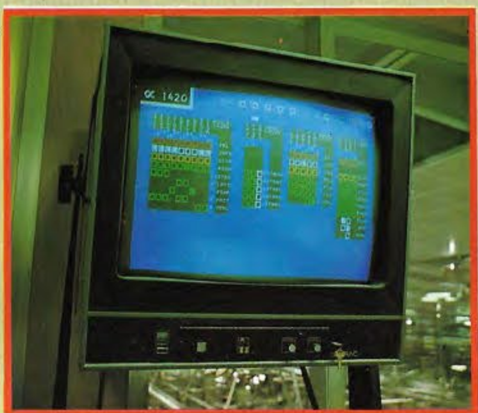
Al lado. Local para el ordeñado en una moderna factoría modelo. Para sobrevivir en un mundo tecnológicamente avanzado, los criadores y productores de leche tienden cada vez más a la gestión computerizada.

Los camiones cisterna de la instalación tienen una capacidad de 225.000 litros

Abajo. El sistema Alfa Laval distribuye a las vacas una cantidad controlada de concentrados alimenticios 4 veces al día, a unas horas preestablecidas. Esto, además de ahorrar tiempo y pesado trabajo manual, se traduce en un fuerte incremento de la producción de leche.



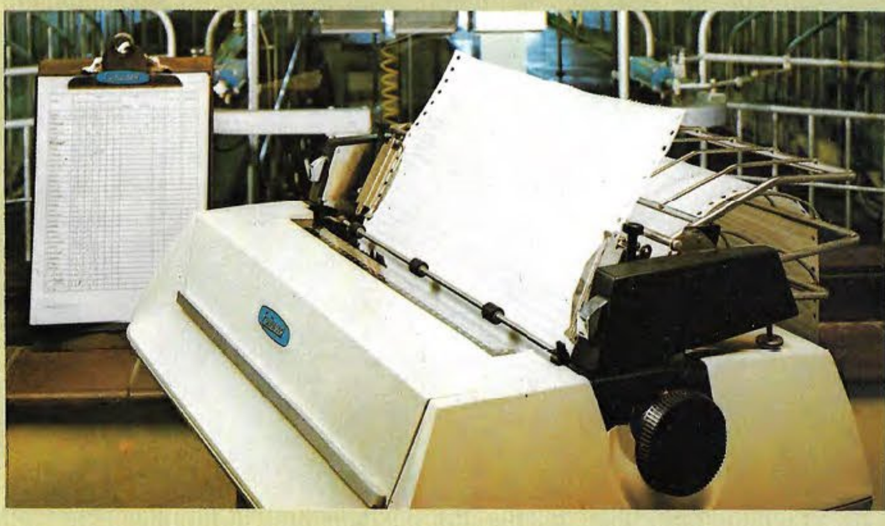
G.R. Roberts



Alfa Laval



R.J. Fullwood & Bland Ltd.



R.J. Fullwood & Bland Ltd.

Arriba. El ordenador proporciona relaciones de los trabajos a realizar, noticias sobre la producción mensual de leche y datos relativos a cada vaca. Al lado. La impresora presenta la documentación completa de las operaciones realizadas.

ría: efectivamente, la producción de leche aumenta; se evitan excesos en la alimentación, se mejora la fecundidad, mientras que para el hombre se reduce la fatiga que comportaba la alimentación manual.

La cantidad de leche efectivamente producida, tanto particularmente como en el conjunto de la factoría se mide mediante un dinamómetro tarado conectado con el recipiente de la leche, mientras que un microprocesador realiza la lectura del peso y lo introduce automáticamente en la memoria del procesador. El control puede ser realizado de forma simultánea en 32 puestos de ordeñado.

Los registros y la cantidad de las informaciones registrables van más allá de la alimentación y la producción de la leche por cada vaca, porque, al ser diversificadas las condiciones y las prestaciones de cada una de las cabezas de ganado, también varían sus necesidades individuales. Con el sistema de gestión de las factorías del Fullwood System, el productor está informado de todos los controles específicos y de los tratamientos que necesita una determinada vaca desde el preciso momento en que entra en el departamento que se le ha asignado en el establo. Cada departamento está equipado con un espía luminoso que se ilumina cuando el sistema responde con un mensaje a la identificación de la vaca que ha ocupado el puesto. Por ejemplo, una vaca puede estar necesitada de curas médicas porque está afectada de mastitis; en tal caso aparecen las instrucciones apropiadas. Podría ser necesario tomar muestras de leche, porque existe la posibilidad de que ésta se deseche directamente si resultara contaminada por los medicamentos suministrados al animal para curarlo de una enfermedad de la que esté afectado. Naturalmente, la función del cuidador no se limita únicamente a responder a instrucciones similares. Si observando una vaca, se da cuenta de que ésta tiene necesidad de determinadas curas, puede introducir el oportuno mensaje en el procesador.

Las necesidades de una factoría para la producción de leche son múltiples; algunas de ellas deben atenderse cada día, mientras que otros trabajos, a pesar de que necesitan ser realizados regularmente, son menos frecuentes. El sistema proporciona al responsable de la instalación una relación diaria de las operaciones a realizar –como, por ejemplo, el tratamiento que debe aplicarse a los animales– y una lista de los acontecimientos menos frecuentes, como los alumbramientos y la ejecución del ordeñado.

Por otra parte, la relación de las operaciones a realizar semanalmente proporciona informaciones sobre los necesarios exámenes relativos a los embarazos en curso y a los controles que deben realizarse sobre los recién nacidos. Una relación de sucesos semanales podría registrar el número de vacas que entran en la vacada o que salen, el número de animales atendidos, la producción de leche calculada independientemente según los usos a que va destinada. Mensualmente, o con mayor frecuencia si fuese necesario, puede realizarse un análisis de las prestaciones del rebaño, en términos de leche producida o desechada, a la luz de elementos como el número de las vacas grávidas o no.

Con el crecimiento de las dimensiones de un criadero, también aumenta la dificultad para el cuidador en la identificación de cada cabeza de ganado y para recordar las características individuales de cada una. Pero el ordenador puede proporcionar instantáneamente un perfil completo de la vida de cada vaca: nombre, progenitores, ascendientes y descendientes, producción durante los días anteriores, son solamente algunos de los datos posibles puestos a disposición por el sistema.

El previsible desarrollo en el futuro prevé una conexión entre el animal y el procesador más estrecha de lo que es posible con el collar, sustituyéndolo con un nuevo aparato que podría ser aplicado de forma permanente a la oreja o al pecho del animal. Además, el procesador podría asumir el control de los cálculos asociados a la producción de la leche, además naturalmente de proporcionar las informaciones que ya elabora actualmente. Por ejemplo, podrían realizarse los oportunos acoplamientos para abrir y cerrar vallas y guiar la vacada por el interior del local de ordeñado.

La entrada del procesador en el sector de la producción lechera ha aportado beneficios análogos a los que ya rinde en las innumerables aplicaciones que tiene en otros sectores de la economía. Sin embargo, en este caso, la madura y compleja experiencia de los agricultores y criadores en la práctica cotidiana ha demostrado ser insustituible.

Sólo gracias a ésta ha sido posible la puesta a punto de paquetes de aplicación que pudiesen suplir algunas necesidades particulares del sector.

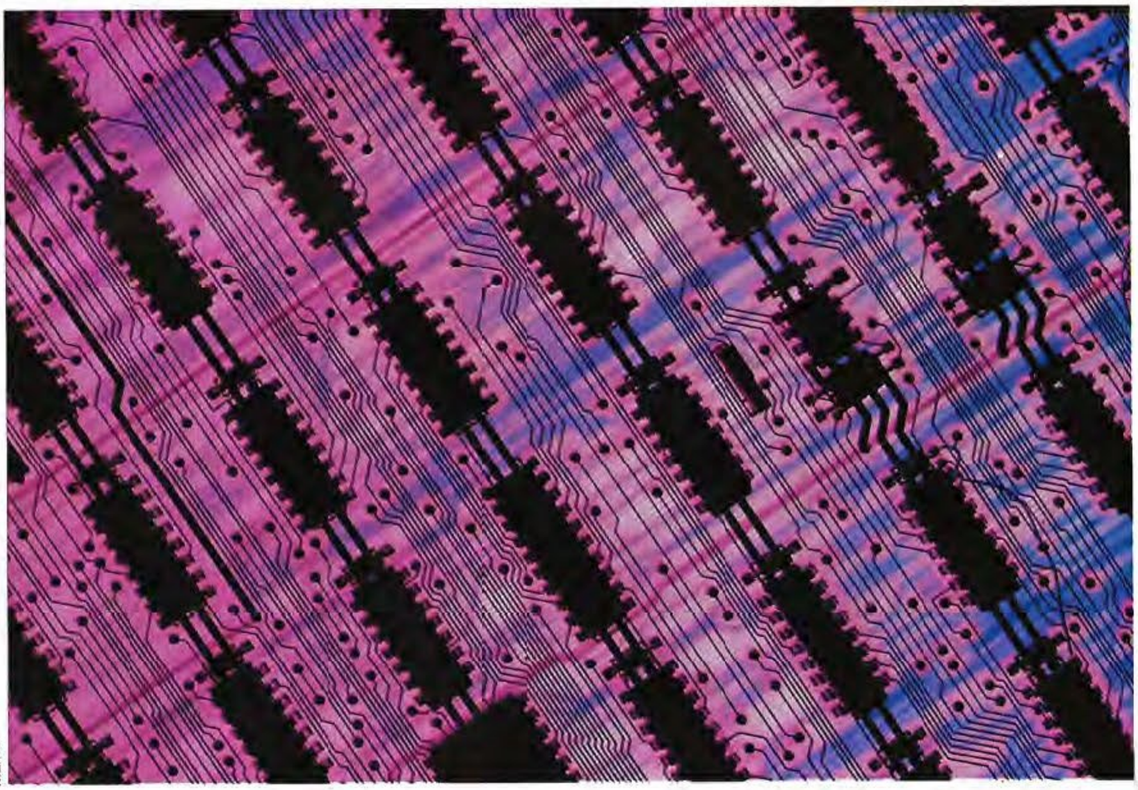
Las matrices

Las variables utilizadas en un programa son nombres simbólicos a que el sistema operativo asocia una memoria. A veces es útil designar con el mismo nombre las memorias; así, cuando debe usarse una tabla (ver el programa de cálculo de disipaciones térmicas en pág. 505), reservando algunas memorias contiguas que tienen el mismo nombre simbólico se pueden utilizar los elementos singulares «direccionándolos» con el siguiente número que tienen en el área de memoria dedicada a ella. En la fase de escritura del programa se deben suministrar al ordenador el nombre del área de memoria reservada a tal efecto y el número de los elementos que la constituyen. La instrucción es

DIM NOMBRE (N)

DIM (dimensión) es el código operativo que indica a la máquina que reserve al programa un área de memoria. Esta se llama NOMBRE y tiene N elementos (N es un número entero). El valor de N se llama **dimensión** de la matriz.
Por ejemplo, la instrucción DIM TABL(20) reserva un área de memoria (**matriz**) de 20 elementos (de hecho, la matriz TABL queda dimensionada en 20) bajo el nombre colectivo de TABL. Para utilizar a uno de estos elementos debe «llamarse» con el número siguiente que tiene en el interior del área de memoria reservada a la matriz; este número se llama **índice del elemento**. Por ejemplo, la instrucción R = 6.8 * TABL(4) toma el valor que hay en la posición 4 del área TABL (el índice es 4), lo multiplica por la constante 6.8 y transfiere el resultado a R.
En el empleo de la instrucción DIM debe prestarse mucha atención a la base de numeración utilizada. El ordenador parte del valor 0, por tanto el primer valor de la tabla TABL es el elemento de índice 0, es decir TABL(0). Para evitar esta incongruencia con nuestro modo de razonar (para el programador es más cómodo referirse al primer elemento como el número 1) debe utilizarse el comando OPTION BASE 1. Los motivos por los que existen estas 2 bases (0 y 1) dependen del mecanismo utilizado para la gestión de las áreas de memoria y se esclarecen en el capítulo dedicado al lenguaje Assembler.
Finalmente, debe tenerse en cuenta que **los elementos que forman parte de una misma matriz deben ser homogéneos**: no es posible

Estos circuitos integrados están dispuestos según una matriz bidimensional.



Marka

memorizar en una misma matriz elementos numéricos de tipo entero y real, ni asignar a una misma matriz cadenas y elementos numéricos. Para la definición del tipo de los elementos de una matriz, valen las reglas enunciadas para las constantes y para las variables.

Memorización de los datos en una matriz

En los programas presentados hasta ahora, cada variable tiene un nombre propio y está memorizada en una área reservada para ella, de una longitud (en número de bytes) que es la del tipo de la propia variable: una variable entera ocupa dos bytes, una cadena emplea un byte por cada carácter, etc. Para las matrices se puede calcular el área ocupada multiplicando el número total de elementos por la longitud (en bytes) de cada uno de ellos. Así, los comandos

```
10 OPTION BASE 1
20 DEFINT A – K
30 DIM AR(3)
```

definen una matriz de nombre AR que posee tres elementos numéricos enteros (la instrucción 20 define como enteras todas las variables cuyo nombre empieza por las letras de A a K, por lo que también la matriz AR se considerará entera); la base es 1 y la ocupación total de memoria es de 6 bytes (dos por cada elemento).

En el Basic existen las direcciones implícitas incluso para las matrices, en el sentido de que pueden utilizarse variables dimensionadas (matriz) aunque no se direccionen explícitamente con el código DIM. En este caso, la asignación de un área de memoria se realiza automáticamente cuando el intérprete encuentra una variable con índice. En el ejemplo anterior, la línea 30 (declaración de variable dimensionada) puede omitirse, ya que el sistema la realizará igualmente durante el curso del programa en el momento en que se haga referencia de la matriz AR(N). Por ejemplo, escribiendo R = 4.7 + AR(3) se crea en la memoria la matriz AR con tres posiciones reservadas a la misma.

Sin embargo, esta forma de dimensionado implícito está sujeta a una limitación que se refiere al número máximo de elementos. El Basic estándar acepta como máximo 10 elementos por cada matriz no declarada explícitamente. La instrucción R = 4.7 + AR(3) se realizará, porque el índice AR es 3, mientras que si fuese R = 4.7 + AR(132) necesitaría la declaración explícita de la dimensión [DIM AR(132)].

La declaración implícita de matriz se utiliza también en las cadenas. Definiendo una variable como cadena puede memorizarse en la misma un número de caracteres variables entre 0 y 128 (en algunos sistemas 256) y el sistema procede automáticamente a direccionar las áreas de memoria. Esta característica del Basic comporta la atribución de un significado particular al código DIM utilizado con una variable de cadena.

Escribiendo DIM A$(3) se reserva un área de memoria que puede contener 3 cadenas diferentes, cada una con un número de caracteres igual al número de las que se han introducido y la ocupación total es la suma de los caracteres de cada cadena. Por ejemplo, el programa

```
10 OPTION BASE 1
20 DIM A$ (3)
30 A$(1) = "Primera cadena"
40 A$(2) = "Segunda"
50 A$(3) = "Ultima"
```

reserva un área de memoria para las tres cadenas [DIM A$(3)], la primera de las cuales contiene 14 caracteres (línea 30; debe considerarse también el espacio entre una palabra y la otra); la segunda, 7 caracteres (línea 40) y la última, 6 caracteres. Por tanto, la ocupación total es de 14 + 7 + 6 = 27 caracteres. Variando una cualquiera de las tres cadenas se tiene una nueva distribución de la memoria. Por ejemplo, poniendo A$(3) = "Ultima cadena", la ocupación total pasa a ser de 34 bytes (se han añadido 7 caracteres). En algún sistema puede imponerse una longitud máxima a cada cadena. Por ejemplo, escribiendo DIM A$(3) [8] se crea un área para 3 cadenas, cada una de las cuales puede contener como máximo ocho caracteres. En este caso, la instrucción 30 del programa anterior produciría un error, porque a A$(1) se asignaría una cadena que contiene más de ocho caracteres. La posibilidad de declarar la longitud máxima de cada cadena es una implantación que sólo está presente en algunos sistemas más evolucionados, para los cuales también existe la posibilidad de definir «subcadenas». El tema volverá a comentarse en el capítulo dedicado a las principales variantes del Basic estándar.

Matrices mono y multidimensional

La declaración DIM AR(3) genera una matriz que incluye tres datos homogéneos; la matriz es **monodimensional** porque se desarrolla según una sola dimensión. Este concepto aparece

más claro estableciendo un paralelo con figuras geométricas. Una línea tiene una sola dimensión; una figura plana tiene dos (longitud y anchura), un sólido posee tres dimensiones (longitud, anchura, y profundidad). Análogamente, una matriz puede tener una sola dimensión, que podemos definir como «longitud», o dos dimensiones, si posee una «longitud» y una «anchura», o bien tres, en el caso de que también exista una «profundidad». Los términos geométricos que se emplean para definir las dimensiones de la matriz no forman parte del vocabulario de la informática. En la programación, los elementos de una matriz de dos dimensiones (bidimensional) se indican como **filas** y **columnas**: una matriz con dos dimensiones, en sustancia, es una tabla dividida en filas y columnas; en el caso tridimensional, la tercera dimensión (correspondiente a la profundidad) puede entenderse como **plano** o **página**. Las primeras dos dimensiones definen un «folio» dividido en filas y columnas, y la tercera dimensión indica qué folio debe considerarse.

Una matriz monodimensional es la estructura más adecuada para albergar una sucesión sencilla de datos, por ejemplo las salidas del balance familiar para cada día del mes. En tal caso, es suficiente dimensionar una matriz de treinta y un elementos, que en la posición 1 deberá contener el total de las salidas del primer día del mes, en la posición 2 las correspondientes al segundo día, y así sucesivamente. Una matriz bidimensional podría utilizarse de forma muy útil para contener una tabla pitagórica, poniendo en el cruce de la fila I con la columna J el producto I * J. A quien quisiera conocer el producto entre los dos números enteros A y B le bastaría con leer el dato contenido en la posición (A, B).

Una matriz tridimensional encuentra más rara vez ocasión de empleo, pero puede resultar igualmente indispensable en varios casos. En la página siguiente se han representado los esquemas de tres matrices que tienen respectivamente 1, 2 y 3 dimensiones; las instrucciones correspondientes de dimensionado son:

— DIM A(4)
  matriz monodimensional
  con cuatro valores
— DIM B(3, 5)
  matriz bidimensional
  con 3 × 5 = 15 valores
  dispuestos en 3 líneas y 5 columnas
— DIM C(3, 3, 5)
  matriz tridimensional
  con 3 × 3 × 5 = 45 valores
  dispuestos en 3 planos,
  3 filas y 5 columnas

El número de las dimensiones de una matriz puede también ser mayor de tres (en este caso no existe un paralelo geométrico) pero la variable estructurada se convierte en complicada para gestionar. En las raras ocasiones en que se hace necesaria una matriz con más de tres dimensiones, conviene dividirla en más matrices, con diferentes nombres y de tres dimensiones como máximo. También debe tenerse presente que algunos intérpretes no aceptan dimensionados más allá de cierto límite.

Instrucciones de asignación para las matrices

Los valores de cada elemento de una variable estructurada pueden ser asignados con métodos análogos a los utilizados para las variables no dimensionadas: asignación directa con el símbolo = y asignación con una instrucción DATA. Por ejemplo, el programa

```
10 OPTION BASE 1
20 'La numeración parte de 1
30 DIM A(4)
40 A(1) = 7:A(2) = 1:A(3) = 3.5:A(4) = 40
```

asigna a cada elemento de la variable A(4) los valores indicados en la pág. 523. El mismo resultado puede conseguirse escribiendo:

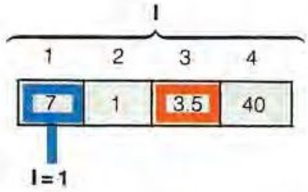
```
10 OPTION BASE 1
20 'La numeración parte de 1
30 RESTORE 70
40 FOR I = 1 TO 4
50 READ A(I)
60 NEXT I
70 DATA 7, 1, 3.5, 40
```

El bucle de las instrucciones 40, 50 y 60 transfiere cada vez uno de los valores contenidos en el DATA (línea 70) en la posición I de la matriz A(4). De este modo, para I = 1, el primer valor del DATA(7) se deposita en la primera posición de la matriz [posición 1; por tanto A(1) = 7], para I = 2, el segundo valor del DATA se transfiere a la segunda posición de A[A(2) = 1], etc.

En las páginas 524-525 se indican los diagramas de flujo para la asignación de los valores a

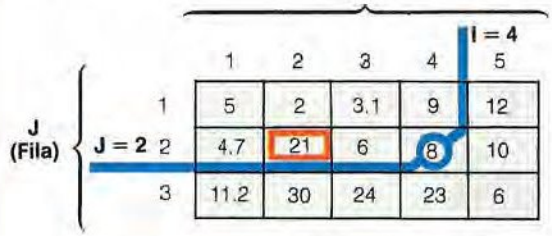
ALGUNOS TIPOS DE VARIABLE ESTRUCTURADA (MATRIZ)

Una dimension



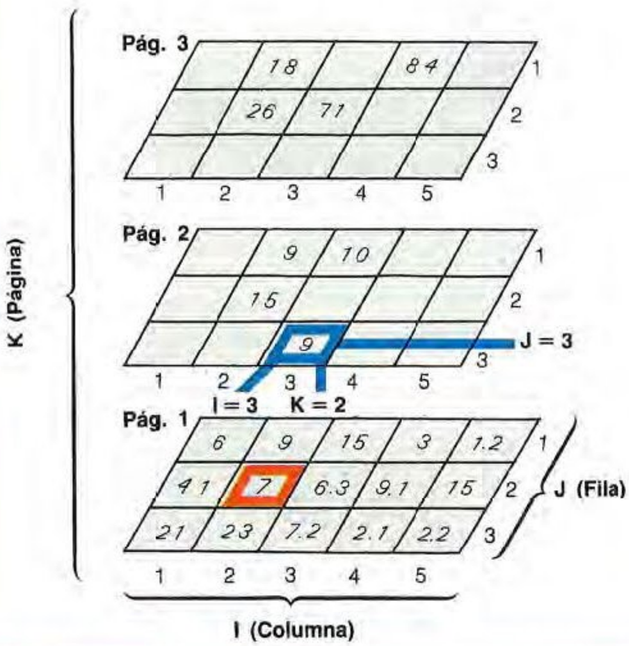
DIM A(4)
Matriz
monodimensional
Ejemplo:
A(3) = 3.5
Poniendo I = 1 ⇒ A(I) = 7

Dos dimensiones



DIM B(3,5)
Matriz bidimensional
Ejemplo:
A(2,2) = 21
Poniendo J = 2, I = 4 ⇒ A(J, I) = 8

Tres dimensiones

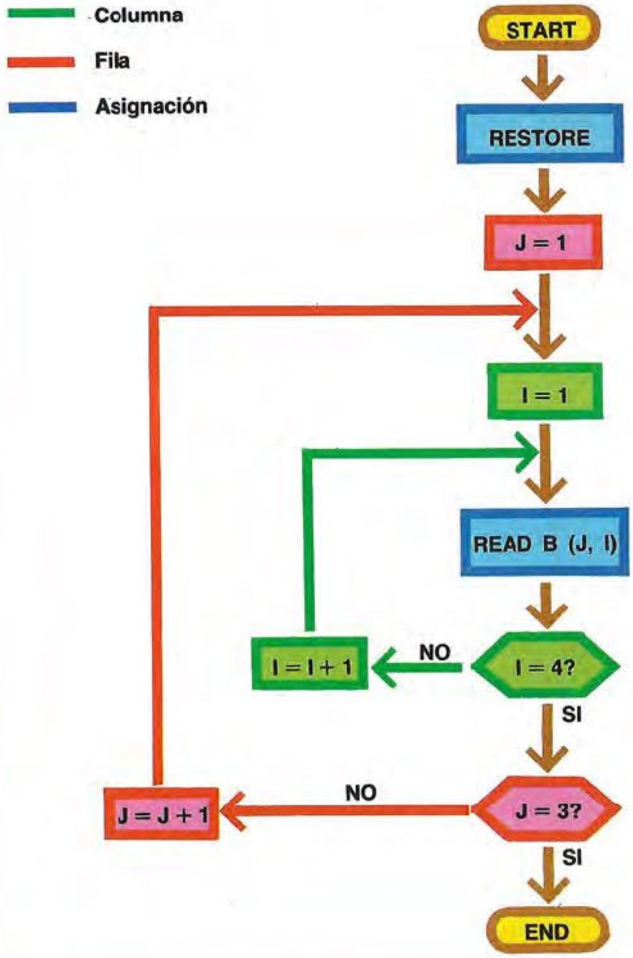


DIM A(3,3,5)
Matriz
tridimensional

La primera dimensión puede entenderse como el número de página; las otras dos como filas y columnas

Ejemplo:
A(1,2,2) = 7
Poniendo I = 3, J = 3, K = 2
⇒ A(K,J,I) = 9

ASIGNACION POR MATRIZ BIDIMENSIONAL (DIAGRAMA DE FLUJO)



dos variables (matriz) de más dimensiones (los listados se indican en las págs. 526 y 527).

La asignación de valores con los DATA puede ser utilizada también por las cadenas, de modo similar a lo que se ha visto para los valores numéricos. Por ejemplo, el programa

```
10 OPTION BASE 1
20 DEFINT I-N
30 DIM A$(3)
40 RESTORE
50 FOR I = 1 TO 3
60 READ A$(1)
70 NEXT I
80 DATA "Primera fila"
90 DATA "Segunda"
100 DATA "Ultima"
```

asigna a las tres cadenas contenidas en A$ los valores «Primera fila», «Segunda», «Ultima». Un ejemplo de aplicación se tiene en la generación de los mensajes de error. En la ejecución de un programa se producen errores, que pueden ser recuperados corrigiendo oportunamente los datos. Ello suele ocurrir en las fases de introducción, cuando el usuario proporciona a la máquina los parámetros para la elaboración. Un buen diagnóstico con mensajes que indi-

ASIGNACION POR MATRIZ TRIDIMENSIONAL (DIAGRAMA DE FLUJO)



ASIGNACION POR MATRIZ BIDIMENSIONAL (LISTADO)

```
10 ' ** PROGRAMA : MATRIZ BIDIMENSIONAL **
20 '
30 '                 FILE = ARRBID
40 '
50 OPTION BASE 1
55 RESTORE 110
60 FOR J=1 TO 3
70 FOR I=1 TO 4
80 READ A (J,I)
90 NEXT I
100 NEXT J
110 DATA 11,12,13,14,21,22,23,24,31,32,33,34
120'
130 FOR J=1 TO 3
140 LPRINT "FILA : ";J
150 FOR I=1 TO 4
160 LPRINT A (J,I);
170 NEXT I
180 LPRINT          ' ESPACIADO
190 NEXT J
200 END
```

```
FILA:   1
 11   12   13   14
FILA:  2
 21   22   23   24
FILA:  3
 31   32   33   34
```

ASIGNACION POR MATRIZ TRIDIMENSIONAL (LISTADO)

```
10 ' ** PROGRAMA : MATRIZ TRIDIMENSIONAL **
20 '
30 '                 FILE = ARRTRI
40 '
50 OPTION BASE 1
60 RESTORE 140
70 FOR K=1 TO 3
80 FOR J=1 TO 3
90 FOR I=1 TO 4
100 READ A(K,J,I)
110 NEXT I
120 NEXT J
130 NEXT K
140 DATA 11,12,13,14,21,22,23,24,31,32,33,34
150 DATA 51,52,53,54,61,62,63,64,71,72,73,74
160 DATA 41,42,43,44,81,82,83,84,91,92,93,94
170 FOR K=1 TO 3
180 LPRINT "PAGINA: ";K: FOR J=1 TO 3:LPRINT "FILA: ";J
190 FOR I=1 TO 4
200 LPRINT A (K,J,I);
210 NEXT I
220 LPRINT
230 NEXT J
240 LPRINT
250 NEXT K
260 END
```

```
PAGINA:   1
FILA:   1
 11 12 13 14
FILA:   2
 21 22 23 24
FILA:   3
 31 32 33 34

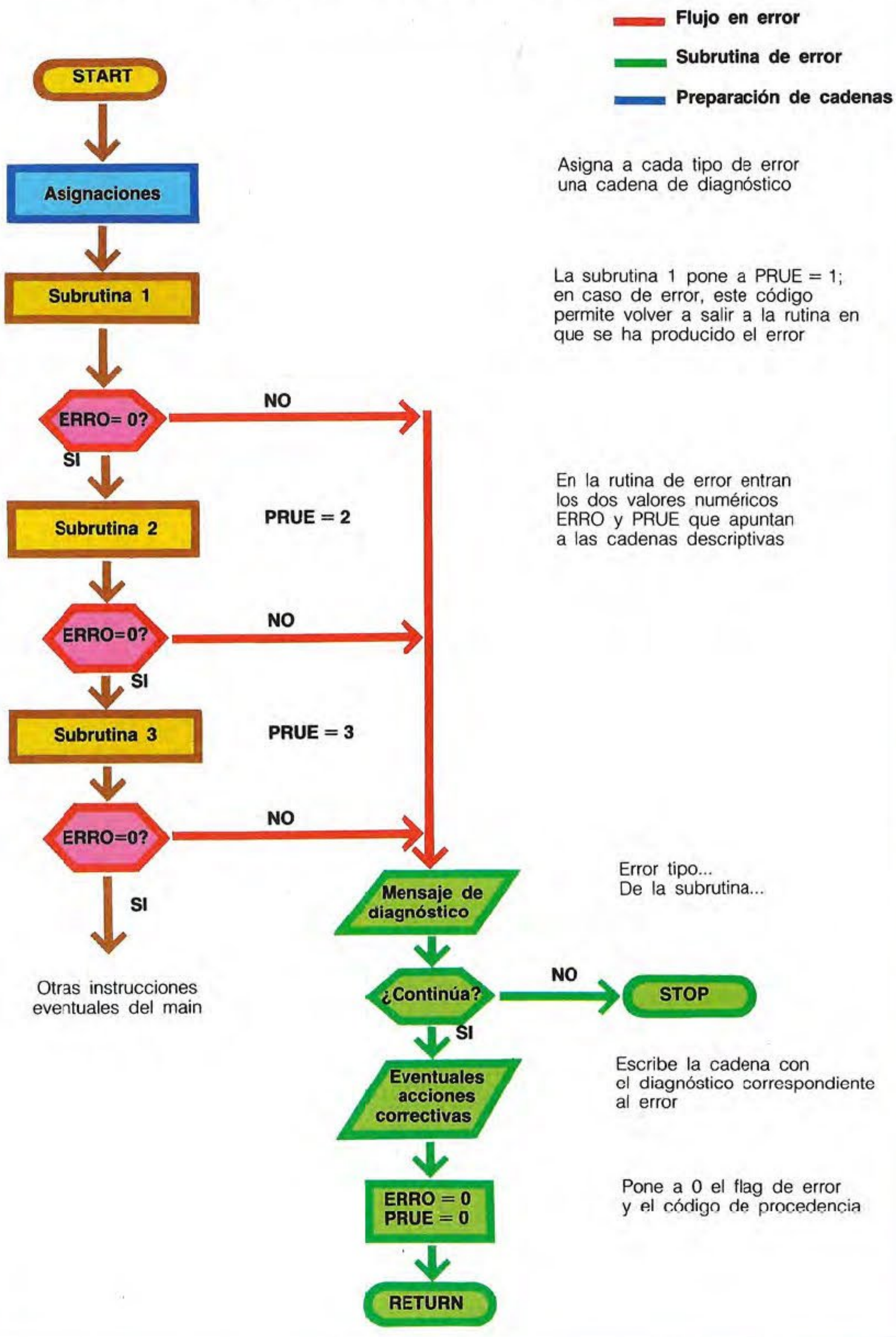
PAGINA:   2
FILA:   1
 51 52 53 54
FILA:   2
 61 62 63 64
FILA:   3
 71 72 73 74

PAGINA:   3
FILA:   1
 41 42 43 44
FILA:   2
 81 82 83 84
FILA:   3
 91 92 93 94
```

quen claramente las operaciones a realizar es esencial para la funcionalidad de los programas. El diagnóstico y la consiguiente escritura que deberán aparecer en la pantalla pueden encontrarse en cualquier punto del programa, pero de este modo se hace difícil la gestión. El procedimiento más correcto consiste en reagrupar los mensajes de diagnóstico en una subrutina única. Al verificarse un error, se llama esta subrutina, en la que el código numérico del error indica el tipo de mensaje que debe presentarse en pantalla. En caso de error, además del mensaje para el usuario, es útil que se presenten en pantalla unas siglas que indiquen qué tipo de error se ha cometido: el conocimiento de estas siglas proporcionará al programador unas útiles indicaciones y lo indicará en las operaciones de ajuste del programa. En la página 528 se presenta el esquema lógico de un programa estructurado de este modo. Las funciones a desarrollar están repartidas en tres subrutinas, con las instrucciones de control y de validación de los datos utilizados. Si éstos no están en los límites previstos se detecta un error, y a continuación en la variable ERRO se escribe un valor numérico que indica el tipo de error que se está cometiendo, mientras que en la variable PRUE se escribe el número correspondiente a la subrutina que ha levantado el flag de error.

Por ejemplo, los valores ERRO = 4 y PRUE = 2 indican que se ha verificado un error de tipo 4 en la subrutina número 2. Los mensajes de diagnóstico y los nombres de las subrutinas están preparados en el módulo de asignación.

PROGRAMA CON LOGICA DE CONTROL DE ERRORES



En esta página se ha incluido el listado de un programa que simula la lógica expuesta, pidiendo al usuario qué valor quiere generar introduciendo la salida correspondiente.

Aplicaciones de las matrices

Las ocasiones de empleo de las variables estructuradas en los programas de aplicación son innumerables. La organización de un cierto número de datos en una matriz optimiza al mismo tiempo la fase de introducción y la ocupación de la memoria, y simplifica notablemente el programa que la utiliza, haciéndolo en definitiva más legible. En este párrafo ilustraremos dos aplicaciones típicas.

La primera contemplará el empleo de las matrices en un programa estructurado para la ejecución de cálculos sobre variables con un número cualquiera de cifras significativas. La segunda dará ocasión para introducir algunos conceptos de estadística aplicada que deberán formar parte del bagaje técnico de cualquier programador. Su conocimiento todavía no es indispensable para la prosecución del estudio.

PROGRAMA DE SIMULACION DE ERRORES

```
10 ' ** PROGRAMA DE SIMULACION DE ERRORES **
20 '
30 '                MAIN
40 '           FILE = SIMERR
50 '
60 ' DECLARACIONES Y ASIGNACIONES
70 OPTION BASE 1
80 DEFINT A-R              ' Las variables cuyos nombres empiezan con letras
90 '                         comprendidas entre A y R son enteras.
100 DIM D$ (4)             ' Las cuatro cadenas D$ contienen las descripciones
102 '                        de los diagnósticos.
110 DIM R$ (3)             ' Las tres cadenas R$ contienen, por extensión,
112 '                        los nombres de las subrutinas.
120 RESTORE 190            ' Apunta al primer DATA (diagnósticos)
130 FOR I=1 TO 4           ' Lectura de la cadena D$
140 READ D$ (I)
150 NEXT I
160 '
170 '** Los DATA que siguen contienen algunos diagnósticos cualesquiera,
172 '    a título de ejemplo.
180 '
190 DATA "Error de lectura o escritura en disco - "
200 DATA "Introducción de datos no válidos - "
210 DATA "Elección de una opción no prevista - "
220 DATA "Se ha introducido una palabra de orden no reconocida - "
230 '
240 RESTORE 320            ' Apunta al segundo DATA (nombres de las Subrutinas)
250 FOR I = 1 TO 3         ' Lectura de las cadenas R$
260 READ R$ (I)
270 NEXT I
280 '
290 ' ** Los nombres son a título de ejemplo. En realidad, las Subrutinas no
300 '      realizan ninguna función.
310 '
320 DATA "Primera Subrutina"
330 DATA "Cálculo de intereses"
340 DATA "Subrutina de prueba"
350 '
360 ' ** Las instrucciones RESTORE en las filas 120 y 140 no son
362 '      estrictamente necesarias.
370 '
380 ERRO=0: PRUE=0            ' Pone a cero los flags
390  GOSUB 1000                ' Llama a la primera subrutina
400  IF ERRO < > 0 THEN GOSUB 9000 ' SI HAY ERROR LLAMA LA 9000
410  GOSUB 2000                ' Segunda Subrutina
420  IF ERRO < > THEN GOSUB 9000
430  GOSUB 3000               'Tercera Subrutina
440  IF ERRO < > 0 THEN GOSUB 9000
480  ' **. FIN DE PROGRAMA
490  '
500  '
```

```
1000 ' ** ENTRADA A LA PRIMERA SUBRUTINA
1010 '    Pide qué error se quiere simular y retorna al MAIN
1020 PRUE=1                  FLAG QUE INDICA LA PROCEDENCIA
1030 INPUT  "Introducir el código del error a simular     ";ERRO
1040 RETURN
1050 '  Las subrutinas 2000 y 3000 son idénticas. Sólo cambia PRUE
1060 '
2000 ' ** SEGUNDA SUBRUTINA
2010 PRUE=2
2020 INPUT  "Introducir el código del error a simular     "; ERRO

2030 RETURN
3000 '    ** TERCERA SUBRUTINA
3010 PRUE=3
3020 INPUT  "Introducir el código del error a simular     "; ERRO
3030 RETURN
9000 ' ** SUBRUTINA DE ERROR
9010 LPRINT "Error n. : "; ERRO
9020 LPRINT "Procedencia : "; PRUE
9030 '
9040 ' * Los errores previstos están comprendidos entre 1 y 4
9050 '    (Ver instrucciones 190-220)
9060 'IF ERRO > 4 OR ERRO < 1 GOTO 9240     ' Error no previsto
9070 '
9080 '  El valor de ERRO puede utilizarse para seleccionar la
9090 '   correspondiente descripción.
9100 '
9110 ' En general, la descripción es D$ (ERRO)
9120 '
9130 A$=D$ (ERRO)              ' Transfiere a A$ el diagnóstico
9140 A$=A$+" "                 ' Añade un espacio para separar la siguiente
9150 '                           escritura
9160 ' El valor de PRUE se utiliza de forma análoga a ERRO
9170 ' para seleccionar la descripción de la subrutina.
9180 '
9190  IF PRUE > 3 OR PRUE < 1 GOTO 9250     ' Procedencia no prevista
9200  A$=A$+R$ (PRUE)      ' Añade la procedencia a la cadena A$
9210  LPRINT A$                       ' Escribe el diagnóstico completo
9220  ERRO=0 : PRUE=0
9230  RETURN
9240  LPRINT "Error no previsto" : STOP
9250  LPRINT "Procedencia no prevista" : STOP
```

```
Error n. : 1
Procedencia : 1
Error de lectura o escritura en disco - Primera Subrutina
Error n. : 1
Procedencia : 2
Error de lectura o escritura en disco - Cálculo de intereses
Error n. : 1
Procedencia : 3
Error de lectura o escritura en disco - Subrutina de prueba

Error n. : 2
Procedencia : 1
Introducción de datos no válidos - Primera Subrutina
Error n. : 2
Procedencia : 2
Introducción de datos no válidos - Cálculo de intereses
Error n. : 2
Procedencia : 3
Introducción de datos no válidos - Subrutina de prueba
```

```
Error n. : 3
Procedencia : 1
Elección de una opción no prevista - Primera Subrutina
Error n. : 3
Procedencia : 2
Elección de una opción no prevista - Cálculo de intereses
Error n. : 3
Procedencia : 3
Elección de una opción no prevista - Subrutina de prueba

Error n. : 4
Procedencia : 1
Se ha introducido una palabra de orden no reconocida - Primera subrutina
Error n. : 4
Procedencia : 2
Se ha introducido una palabra de orden no reconocida - Cálculo de intereses
Error n. : 4
Proce encia : 3
Se ha introducido una palabra de orden no reconocida - Subrutina de prueba

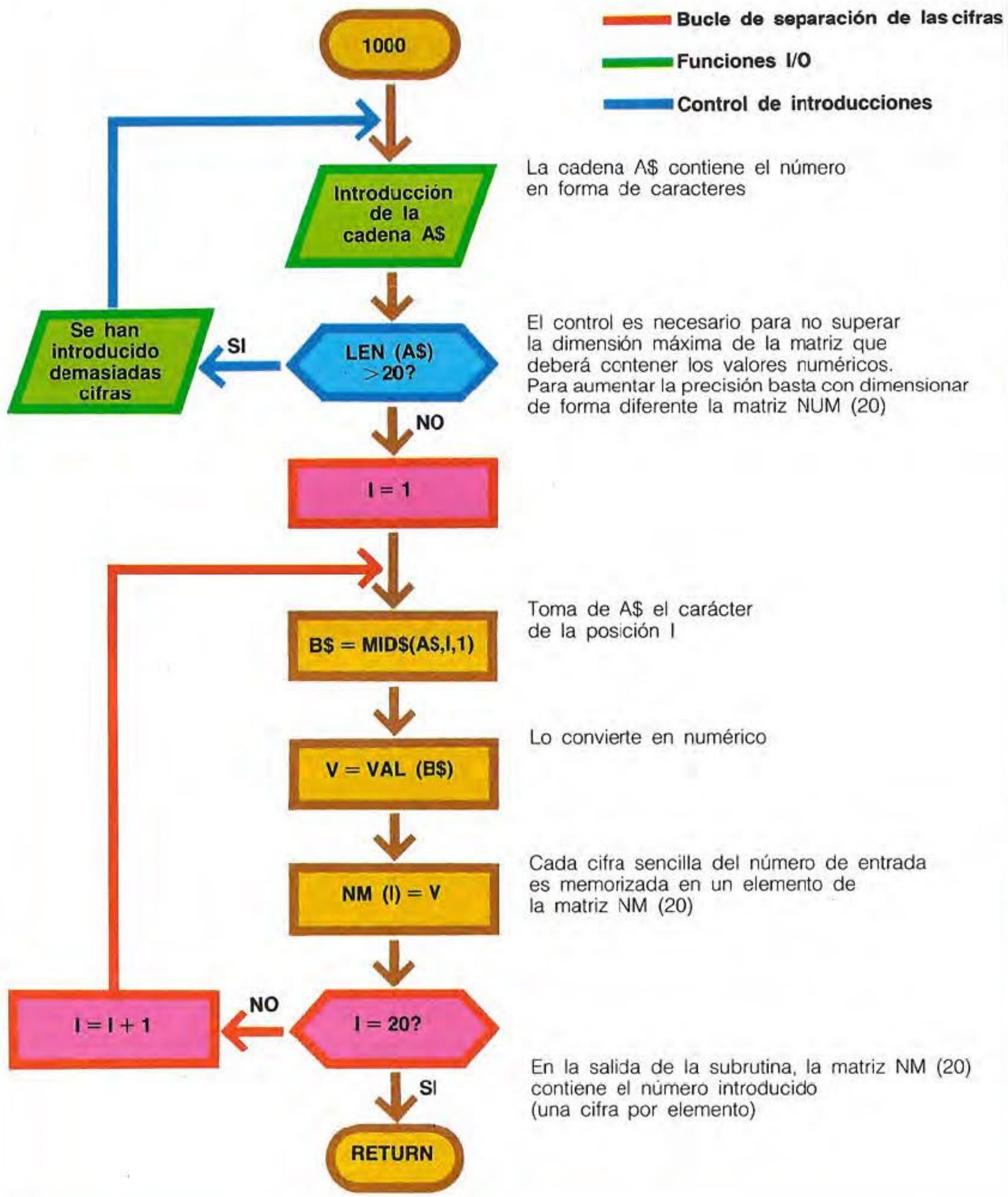
Error n. : 9
Procedencia : 1
Error no previsto
```

Desarrollo de cálculos con un número cualquiera de cifras significativas. En la casi totalidad de las aplicaciones prácticas, la precisión de los cálculos obtenible con los números normales en doble precisión es suficiente para excluir la posibilidad de errores. Sin embargo, pueden verificarse situaciones en que es necesario desarrollar cálculos con una precisión mayor. Los métodos o las soluciones alternativas para la utilización son múltiples y todas implican una cierta dificultad, a veces importante.
El problema de la precisión en los cálculos debe ser afrontado según dos puntos de vista diferentes: el científico y el inherente a aplicaciones para cálculos económicos.
En las aplicaciones científicas no debe presentarse la dificultad de la precisión en los cálculos. Si existe esta dificultad, en la casi totalidad de los casos se debe a errores en los algoritmos o a su uso equivocado. Las aplicaciones científicas describen fenómenos físicos reales (mensurables) y si el desarrollo de un cálculo necesita el uso de precisiones demasiado elevadas, la causa debe buscarse en el modo en que está estructurado el programa, y no en la precisión de la máquina. En el campo económico debe hacerse el razonamiento opuesto: al presentar un balance o cualquier otro documento contable, debe cuadrar hasta la última cifra significativa, cualquiera que sea la magnitud del número. Por ejemplo, decir que la velocidad de un coche es de 100 km/h o de 100.02 km/h, desde el punto de vista físico es exactamente la misma cosa; en cambio, si las dos cifras representan dinero contenido en el cálculo de un balance, se tendrá un error de «cuadratura».
Un método, para no estar sujeto al vínculo de la precisión máxima permitida con la máquina, es el de considerar los números como caracteres, es decir, elementos de una cadena. Así, el número de cifras que es posible tratar aumenta hasta un máximo igual a la longitud máxima, en caracteres, de la cadena (128 o 156).
A continuación ilustramos un método que permite realizar sumas entre números de cualquier magnitud, admitiendo los valores numéricos en forma de cadena. Si como dato numérico quisiéramos introducir en el calculador un número de 20 cifras, éste quedaría truncado en una cantidad de cifras igual a la de la precisión de la máquina. Presentando el mismo número como cadena, ésta entra directa e íntegramente como si fuese cualquier otra cadena de 20 caracteres. Luego se toman los caracteres simples de la cadena y se transfieren a una matriz numérica*. Así, la representación numérica ya no se obtiene como valor en notación decimal, (como un número único), sino como un conjunto de cifras separadas, cada una en una posición de me-

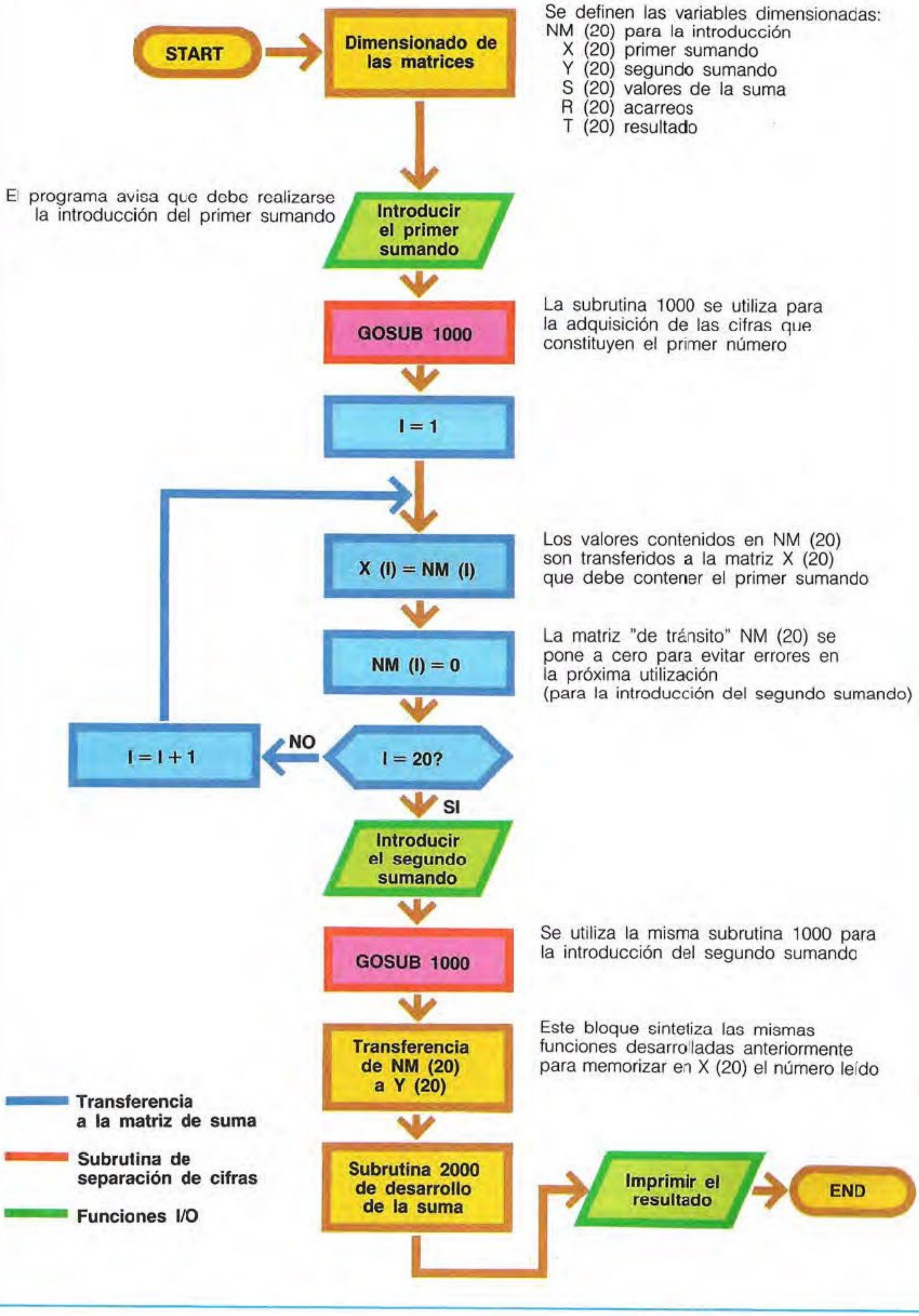
* A partir de ahora, los listados de los programas podrán no incluir la instrucción OPTION BASE 1, ya que el valor 1 es asumido automáticamente por omisión en muchos sistemas operativos. Sin embargo, se advierte que esta circunstancia debe verificarse previamente en la máquina particular.

moria distinta. La máquina ya no debe memorizar el número como dato único, con los consiguientes problemas de precisión, sino que lo representa como una matriz, cuyos elementos contienen una sola cifra entre 0 y 9. Abajo se indica el diagrama de flujo de la subrutina que permite la memorización de números formados hasta con 20 cifras; utilizándola 2 veces se adquieren los 2 números a sumar con los cuales deberá operarse. En la pág. 533 aparece el diagrama de flujo del main que realiza la suma entre dos datos numéricos.

SUBRUTINA DE MEMORIZACION DE DATOS EN PRECISION EXTENDIDA



SUMA DE DATOS NUMERICOS EN PRECISION EXTENDIDA



Se definen las variables dimensionadas:
NM (20) para la introducción
X (20) primer sumando
Y (20) segundo sumando
S (20) valores de la suma
R (20) acarreos
T (20) resultado

El programa avisa que debe realizarse la introducción del primer sumando

La subrutina 1000 se utiliza para la adquisición de las cifras que constituyen el primer número

Los valores contenidos en NM (20) son transferidos a la matriz X (20) que debe contener el primer sumando

La matriz "de tránsito" NM (20) se pone a cero para evitar errores en la próxima utilización (para la introducción del segundo sumando)

Se utiliza la misma subrutina 1000 para la introducción del segundo sumando

Este bloque sintetiza las mismas funciones desarrolladas anteriormente para memorizar en X (20) el número leído

El cálculo de la suma, elemento por elemento, lo realiza la subrutina 2000, cuyo listado se indica aquí abajo, mientras que en la página siguiente se ha presentado un ejemplo de la lógica seguida. El ejemplo muestra el cálculo de la suma de los dos números 798899 y 4568 según el método utilizado en la subrutina 2000.

EJEMPLO DE CALCULO UTILIZANDO CADENAS DE MATRIZ

```
10 ' FILE = ST1
15 DEFINT A-Z
20 DIM NM(20), X(20), Y(20), S(20), R(20), T(20)
30 PRINT "Introducir primer sumando"
40 GOSUB 1000
45 LPRINT ' VALOR PRIMER    SUMANDO = ";A$
50 FOR I=1 TO 20
60 X(I)=NM(I)
70 NM(I)=0
80 NEXT I
90 PRINT "Introducir segundo sumando"
100 GOSUB 1000
105 LPRINT "VALOR SEGUNDO SUMANDO = ";A$
110 FOR I=1 TO 20
120 Y(I)=NM(I)
130 NM(I)=0
140 NEXT I
150 GOSUB 2000
160 PRINT "Resultado "
170 FOR I=1 TO 20:PRINT S(I);:NEXT I
175 LPRINT
176 LPRINT "RESULTADO :"
177 FOR I=1 TO 20:LPRINT S(I);:NEXT I
180 END
1000 ' ** SUBRUTINA DE INTRODUCCION **
1010 INPUT "Introducir valor";A$
1020 IF LEN(A$) >20 THEN PRINT "Error":GOTO 1010
1030 FOR I=1 TO 20
1040 B$=MID$(A$,I,1)
1050 V=VAL(B$)
1060 NM(I)=V
1070 NEXT I
1080 RETURN
2000 ' ** SUBRUTINA DE CALCULO **
2010 FOR I=20 TO 1 STEP -1
2020 V=X(I)+Y(I)          ' Calcula la suma de los dos elementos en I
2030 IF I=20 GOTO 2050 ' La primera suma no tiene acarreo
2040 V=V+R (I+1)          ' Suma el acarreo de la vuelta anterior
2050 V1=INT (V/10)
2060 R (I)=V1             ' Memoriza el nuevo acarreo (usado en la siguiente vuelta)
2070 V2=V-V1*10           ' Aisla las unidades que constituyen la cifra
2075 '                      en la posición I del resultado
2080 S(I)=V2              ' Memoriza la cifra en la posición I
2090 NEXT I
2095 RETURN
```

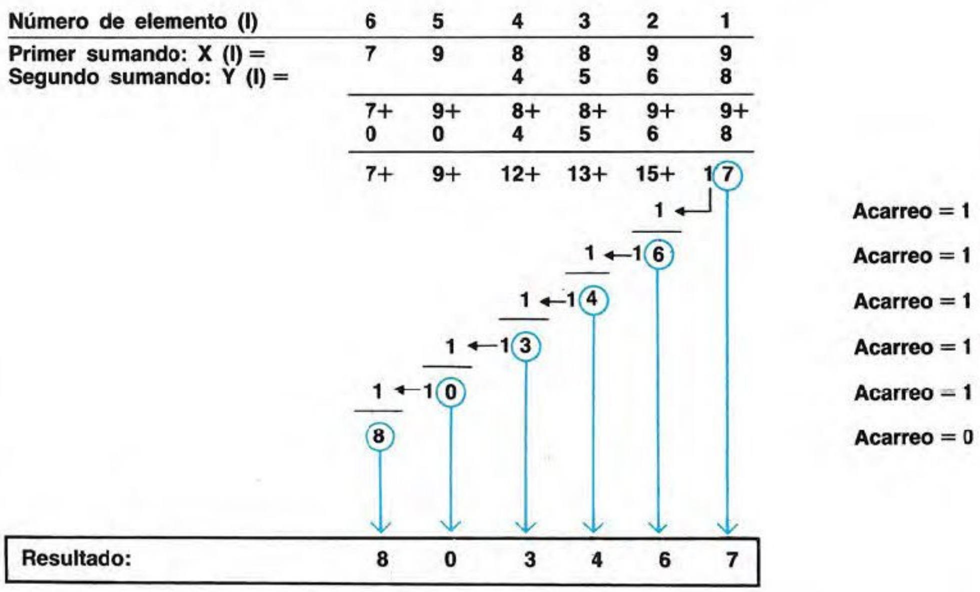
```
VALOR PRIMER SUMANDO  = 01234567899876543210
VALOR SEGUNDO SUMANDO = 09876543210123456789

RESULTADO :
1 1 1 1 1 1 1 1 1 0 9 9 9 9 9 9 9 9 9 9


VALOR PRIMER  SUMANDO = 01234567899876543210
VALOR SEGUNDO SUMANDO = 89764582136987412304

RESULTADO :
9 0 9 9 9 1 5 0 0 3 6 8 6 3 9 5 5 5 1 4
```

SUMA DE ENTEROS EN PRECISION EXTENDIDA



TEST 15

1 / Algunas de las siguientes funciones son erróneas; decir cuáles y por qué.

a) DEF FNA = CHR$(27) + CHR$(10)
b) DEF FNB = C * 2 + 1
c) DEF FNN$ = CHR$(7) + "Valor"
d) DEF FNA$ = "Cadena de prueba" + " = " + 256

2 / Escribir un programa que realice las siguientes funciones:

- Introducción de una cadena y de un valor numérico comprendido entre 0 y 99
- Comprobar si este valor numérico representa en ASCII un carácter alfabético
- En caso afirmativo, buscar este carácter en la cadena (sólo el primero)
- El programa debe trabajar en tanto no se introduzca la cadena "FIN"

3 / Escribir un programa de totalización que realice las siguientes funciones:

a) Lectura de un grupo de datos constituido por
Código = valor numérico comprendido entre 10 y 20 introducido por teclado
Importe unitario = valor numérico comprendido entre 1250 y 5700 introducido por teclado
Cantidad = valor numérico introducido por teclado
Nombre = cadena alfanumérica de 20 caracteres introducida por teclado
El programa debe terminar introduciendo Código = 0

b) Comprobar que el Nombre introducido vuelve a entrar después de 10 prefijos (asignados con un DATA)

c) Cálculo del importe total para cada grupo de datos (Importe total = Importe unitario × Cantidad)
d) Suma de la Cantidad y de los Importes introducidos por cada Código
e) Suma del importe total para cada Nombre
f) Impresión de la suma calculada en los dos pasos anteriores cuando se introduce Código = 0

Se aconseja dibujar el diagrama de flujo utilizando las siguientes variables:
CODIGO(11) para los valores de los totales por código
NMT(10) totales por nombre
NOMI$(10) cadenas que contienen los nombres previstos

4 / Escribir una subrutina a insertar en el programa anterior para el cálculo del importe unitario medio sobre la base de los datos introducidos. Se recuerda que la media de N valores viene dada por su suma dividida por N.

5 / El **factorial** de un número N se indica con N! y se obtiene multiplicando entre sí todos los números enteros comprendidos entre 1 y N:
Factorial = N! = 1×2×3...×N
Por ejemplo:
5! = 1×2×3×4×5 = 120
Escribir una subrutina para el cálculo del factorial de un número asignado a la variable NUM. El número debe ser entero y mayor de 2.

6 / Las **combinaciones** de N elementos **de clase K** son todos los posibles agrupamientos que pueden obtenerse con los elementos dados (N) tomados en grupos de K cada vez, sin repeticiones (un elemento no puede estar presente más de una vez en el mismo grupo).
Por ejemplo, con las letras A, B, C (N = 3), tomando dos cada vez (K = 2) se pueden obtener los grupos: AN AC BC
En general, el número de grupos que pueden obtenerse (número de las combinaciones) viene dado por la expresión

$$\left(\begin{array}{c}\text{combinaciones}\\ \text{de N elementos en clase K}\end{array}\right) = \frac{N!}{K!\,(N-K)!}$$

Este valor se indica también de esta manera $\binom{N}{K}$

En el ejemplo anterior (N = 3, K = 2) se tiene:

$$\text{combinaciones} = \frac{3!}{2!(3-2)!} = \frac{1\times2\times3}{1\times2} = 3$$

Escribir un programa para el cálculo del número de las combinaciones que pueden obtenerse con N elementos agrupándolos en grupos de K elementos.

Las soluciones, en las páginas 545, 546 y 547.

Aplicaciones a la estadística. Para describir fenómenos masivos como el total de la población residente en un determinado país o las ventas medias anuales por cliente en una compañía, es necesario tener a disposición un gran número de observaciones de los fenómenos individuales que la componen*.

Elaborar esta enorme cantidad de datos comporta exigencias (como la exactitud y la velocidad de respuesta) que no pueden ser satisfechas con el cálculo manual. Esta es la razón de que un procesador electrónico sea el instrumento indispensable para el estudio de un fenómeno: basta con escribir programas de carácter general que en la entrada acepten un número cualquiera de observaciones y que «programen» el procesador para realizar los cálculos necesarios con el fin de obtener los datos estadísticos que describen el fenómeno.

Como ejemplo incluimos a continuación unos programas para el cálculo de las principales magnitudes que aparecen en la estadística.

Supongamos que tenemos un conjunto de datos observados y que queremos representarlos sintéticamente. Es decir, queremos conocer una característica «global», un valor que, sustituido en los diversos valores observados, no altere las características estadísticas. Se dice en este caso que buscamos una **media**.

Esta magnitud permite rápidas valoraciones globales, facilita las comparaciones y guía en las situaciones de incertidumbre.

Existen varios tipos de medias, que se diferencian según la función característica de los datos a observar (media geométrica, media armónica, media aritmética, mediana, etc.).

La que se considera más normalmente en los cálculos estadísticos es la **media aritmética**, que se calcula como sigue:

$$\text{media aritmética} = \frac{\text{suma de los datos}}{\text{número de los datos}}$$

La media se expresa en las mismas unidades de medida que los datos que se emplean para su cálculo. Para calcular la media de los ingre-

* El resultado de una observación realizada sobre uno de los fenómenos individuales se llama **unidad estadística**; el resultado de una operación realizada sobre la unidad estadística se llama **dato estadístico**.

El sofisticado equipo electrónico del Instituto Central de Estadística italiano (ISTAT).



ISTAT

sos anuales de seis familias

$X_1 = 1.045.000$ ptas/año
$X_2 = 1.170.000$ ptas/año
$X_3 = 1.095.000$ ptas/año
$X_4 = 1.080.000$ ptas/año
$X_5 = 1.230.000$ ptas/año
$X_6 = 1.195.000$ ptas/año

en primer lugar debe calcularse el ingreso total, que es igual a la suma de los seis ingresos,

$$X_1 + X_2 + X_3 + X_4 + X_5 + X_6 = 6.815.000$$

El ingreso medio se obtiene como sigue:

$$\bar{X} = \frac{6.815.000}{6} = 1.135.833{,}33 \text{ ptas/año}$$

Este ingreso medio es el que cada familia hubiera tenido si el ingreso total de 6.815.000 ptas se hubiese repartido en partes iguales entre las seis familias.
En la página siguiente se ha incluido un programa que calcula la media aritmética de un cierto número de datos (MAX) introducidos por consola. El programa utiliza una matriz dimensionada en 100, lo que significa que podrá calcular como máximo la media de 100 datos.

Los diversos tipos de medias se obtienen combinando los datos observados de diferente manera. Pero existe otro tipo de media que se calcula considerando solo la posición de los datos. Este valor es la **mediana**, que se identifica con la media de los dos términos centrales de la serie de n datos (si n es par), o con el término central (si n es impar).
Veamos con un ejemplo por qué puede ser útil calcular la mediana de un conjunto de observaciones en lugar de la media. Se debe suministrar pan a cinco establecimientos de géneros alimentarios (A, B, C, D, E) situados en diferentes zonas de la ciudad, pero cercanos a una calle que la atraviesa. Los establecimientos distan de la calle lo que se indica:

establecimiento	A B C D E
distancia (km)	1 2 3 8 11

Se debe encontrar el punto más cercano posible a los cinco establecimientos en donde poder construir un nuevo horno. El puesto mejor será aquel que, sumando las distancias absolutas entre el horno y todos los establecimientos, dé el menor resultado, y este último será el valor de la mediana. En este caso, los datos son en número impar, y el valor de la mediana es el elemento central (3 km, punto C).

Introduzcamos una nueva magnitud estadística: queremos saber cómo varían los precios de cierto producto alrededor del valor de su media; saber para ello si los precios son más o menos cercanos al precio medio.
Para caracterizar esta variabilidad se emplea un índice llamado **variancia**, que se calcula así:

$$\text{variancia} = \frac{\text{suma de las desviaciones al cuadrado}}{\text{número de datos}}$$

donde la desviación es la diferencia entre el valor de cada dato y la media. La variancia también puede expresarse en la forma:

$$\text{variancia} = \frac{\text{suma de los datos al cuadrado}}{\text{número de datos}} - \text{media de los datos al cuadrado}$$

expresión esta, que simplifica el cálculo. Volvamos al ejemplo y consideremos los precios de las calidades de un producto (por ejemplo miel):

Calidad	**Precio (ptas/kg)**
A	$X_1 = 610$
B	$X_2 = 650$
C	$X_3 = 680$
D	$X_4 = 700$
E	$X_5 = 745$
F	$X_6 = 800$
G	$X_7 = 820$

La suma de los datos es 5005
La media de los precios es

$$\frac{5005}{7} = 715$$

Para el cálculo de la variancia se tiene:

$X_1^2 = 372\ 100$
$X_2^2 = 422\ 500$
$X_3^2 = 462\ 400$
$X_4^2 = 490\ 000$
$X_5^2 = 555\ 025$
$X_6^2 = 640\ 000$
$X_7^2 = 672\ 400$

PROGRAMA PARA EL CALCULO DE MEDIAS

```
10 ' **  PROGRAMA PARA FL CALCULO DE MEDIAS
20 '
30 '       FILE : STAT1
35 B$="£££.£"              ' Formato de impresión
40 '
45 DEFDBL A-H
46 DEFINT I-N
47 DEFDBL O-Z
50 DIM A (100)            Matriz para la memorización de las observaciones
60 '                      máximo 100 valores
70 INPUT  "  Sobre cuántos datos se desea el cálculo ";MAX
80 IF MAX=0  OR MAX >100 THEN PRINT " ** ERROR ** ":GOTO 70
90 ' * Introducción de los valores
100 FOR I=1 TO MAX
110 PRINT " Dato Num.      :";I
120 INPUT " Valor          :"A (I)
130 NEXT I
140 ' ** CALCULO DEL TOTAL**
150 T=0£                              ' Puesta a cero de la suma de datos
160 FOR I=1  TO MAX
170 T=T+A(I)                          ' Acumula la suma de datos en T
180 NEXT I
190 ' ** Las líneas 160 a 180 pueden eliminarse
200       simplemente calculando la suma durante la introducción
210       es decir, introduciendo la instrucción T=T+A(1) en la
220 '     posición 125
230 TMEDIA=T/MAX                    ' El nombre usado para la media debe
231 '                                 empezar con una de las letras asignadas a
232 '                                 la doble precisión
233 '
234 '
240 '
250 ' ** Impresora
260 '
270 LPRINT "Datos:"
280 '  Los datos se imprimen en número de 10 por línea
290 '
300 FOR I=1 TO MAX STEP 10
305 I1=I+9
310 FOR J=I  TO I1
320 LPRINT USING B$, A (J);
330 NEXT J
340 LPRINT
350 NEXT I
360 '
365 LPRINT : LPRINT
370 LPRINT " Media : ";TMEDIA
380 '
390 INPUT "Continúa (SI/NO) :"; RESP$
400 IF RESP$= "SI" GOTO 70
410 END
```

```
Datos
10.2 15.0 15.2 14.4 13.2 10.0 11.0 10.9 11.6 12.0
12.8 12.4 15.9 16.7 16.1 16.8 11.2 10.0 12.0 13.0
14.0 15.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0

Media  :  13.15545454545455
```

La suma es 3 614 425; por tanto:

$$\frac{\text{suma de los datos elevado al cuadrado}}{\text{número de datos}} - (\text{media})^2 = $$

$$= \frac{3\,614\,425}{7} - 511\,225 = 516\,346.42 -$$

$$- 511\,225 = 5\,121.42$$

La inteligencia artificial

Desde un cierto punto de vista, el calculador es análogo a un cerebro: ambos elaboran informaciones procedentes de sus órganos sensoriales externos, y ambos los restituyen elaborados a otros órganos externos. Puesto que los procesadores son máquinas construidas por el hombre, basados en componentes siempre diferentes y más perfeccionados, pero que obedecen siempre a la lógica binaria del álgebra de Boole, he aquí que la analogía entre el cerebro humano, una «caja negra» de la que se sabe bien poco porque tiene demasiados componentes (cerca de 10^{11} neuronas, conectadas entre sí de manera aparentemente caótica), y un procesador, del que se sabe todo porque así ha sido construido, ha puesto en vigor una escuela de pensamiento que dice sustancialmente: estudiemos el funcionamiento del calculador, o procesador de informaciones, y comprenderemos el funcionamiento del cerebro. Éste es visto como un procesador particular, construido con neuronas en lugar de transistores, pero basado en la misma lógica binaria. Cada neurona de hecho es un sistema biestable, exactamente como un circuito flip-flop. Este punto de vista está claramente formulado por H.A. Simon y A. Newell, dos de sus más influyentes valedores:

«Existe un número creciente de indicios que hacen pensar que los procesos informativos elementales usados por el cerebro humano en el pensamiento son muy similares a un subconjunto de procesos informativos elementales que están incorporados en los códigos de instrucciones de los calculadores actuales. Como consecuencia, se ha encontrado que es posible verificar las teorías sobre la elaboración de informaciones por parte de seres humanos formulando estas teorías como programas de calculadores –organizaciones de los procesos elementales informativos– y examinando los output de los calculadores programados así. Este procedimiento no comporta ninguna similitud entre el calculador y el cerebro a nivel de hardware, sino sólo una similitud en su respectiva capacidad de realizar y organizar procesos elementales de información. A partir de esta hipótesis ha nacido una fructífera colaboración entre la investigación de la "inteligencia artificial" con el fin de ampliar la capacidad de los procesadores, y la investigación sobre la psicología de los procesos cognoscitivos humanos».

Partiendo de estas premisas, Simon y Newell llegan a un modelo del cerebro humano como solución de problemas: el General Problem Solver. Se trata de un programa que tiende a simular el comportamiento de un ser humano que se encuentra frente a un problema del que desconoce el método para llegar a la solución, como por ejemplo en una partida de ajedrez. Pero ¿basta esto para concluir que el problema de la inteligencia artificial está resuelto, al menos teóricamente, y que, dado un calculador bastante potente, podremos simular el comportamiento de un ser humano en cualquier circunstancia?

Es necesario evitar llegar a conclusiones prematuras. Según D. Marr y H.K. Nishihara, del Artificial Intelligence Laboratory del MIT, «la inteligencia artificial es (o debería ser) el estudio de los problemas de elaboración de la información que tienen sus raíces características en cualquier aspecto de la elaboración biológica de las informaciones». Hemos accedido a la solución de problemas y a los procesos cognoscitivos; otro campo de vital importancia para los seres vivientes, ya que de este depende especialmente su propia existencia, y también el reconocimiento de las formas, o pattern recognition, *según el término inglés de uso corriente. Se piensa en el animal que debe reconocer la presa (o el depredador), en el hombre que debe identificar una cara o una voz, o bien distinguir un misil balístico de una bandada de gansos sobre una pantalla de radar... Son estos procesos de reconocimiento en los que el cerebro animal tiene una habilidad superior con respecto al procesador, que a su vez lo bate por la velocidad con que elabora los datos numéricos. Probablemente, esto es debido a la naturaleza eminentemente paralela y fuertemente redundante de la elaboración de las informaciones del cerebro, respecto a la elaboración eminentemente en serie del calculador, al menos en el «clásico». No fue por casualidad que los primeros procesadores de estructura paralela (los* parallel processor*) fueron desarrollados en gran parte dentro de programas de reconocimiento de imágenes (ópticas y radar) en tiempo real, con finalidades científicas e incluso militares.*

Por tanto, no es por casualidad que uno de los múltiples desarrollos más avanzados, proyectado y construido en Italia, el EMMA (Elaboratore Multi Mini Associativo) de la Elettronica San Giorgio - Elsag S.p.A. de Génova, ha sido desarrollado en relación a los problemas de recono-

cimiento de caracteres ópticos (OCR: Optical Character Recognition) para la lectura automatizada de direcciones postales en el SARI (Sistema Automatico Riconoscimento Indirizzi), desarrollado por Elsag para el servicio de Correos italiano, después adoptado también en Francia y experimentado en Norteamérica.

El EMMA es un multiprocesador de estructura jerárquica, es decir, organizado en «familias» constituidas cada una por un gran número de módulos de base que pertenecen a tres variedades fundamentales únicas: unidad de cálculo, unidad de memoria y unidad de coordinación de los intercambios de datos. Cada unidad está realizada físicamente sobre una tarjeta de circuito impreso que contiene los circuitos integrados y los componentes pasivos; las tarjetas son conectadas entre sí mediante un «bus» que permite el intercambio de los datos, y todo conjunto es modular y expandible según las exigencias. La estructura jerárquica del EMMA es conceptualmente similar a la de uno de los más avanzados multiprocesadores científicos norteamericanos, el Cm de la Carnegie -Mellon University, pero, a diferencia de éste, el EMMA es un producto industrial. Sin embargo, aquí no entraremos en problemas de estructura de hardware y software de los multiprocesadores. Sólo queremos poner de manifiesto que el hecho de tener que afrontar un problema de inteligencia artificial, según la definición de Marr y Nishihara, ha conducido al desarrollo de un procesador de tipo particular, con características de fiabilidad, tolerancia a las averías y redundancia que recuerdan algunas de las características del sistema nervioso central de los animales superiores y, en particular, el del hombre.

Otro problema de pattern recognition sobre el que ya se ha realizado mucho trabajo y del que empiezan a aparecer las primeras soluciones industriales es el del reconocimiento de la voz; un problema correlacionado, que también tiene posibilidades judiciales, es el de la identificación del orador. Aquí entramos en un campo de ciencia ficción, el del diálogo directo hombre-máquina (es obligada la cita del HAL, el ordenador parlante de «2001 Odisea del espacio», el más bello filme de Kubrick). Efectivamente, si pudiésemos programar con la voz un ordenador, usando el lenguaje natural o su subconjunto, la máquina perdería gran parte de su naturaleza de... máquina; si después ésta, a su vez, respondiese con la voz, ya no sería una máquina. Pero ¿en qué punto están las cosas? En los últimos cinco o seis años se ha dado un impulso a los estudios sobre el reconocimiento vocal, debido a un progreso en el análisis acústico de la señal vocal y en su representación paramétrica, a un mayor conocimiento de los fenómenos fonológicos (interacción entre fenómenos cercanos e intercambio en su realización acústica) y lingüísticos, a la toma de conciencia de la importancia del contexto sobre la comprensión de las palabras sencillas y al progreso general de los procesadores.

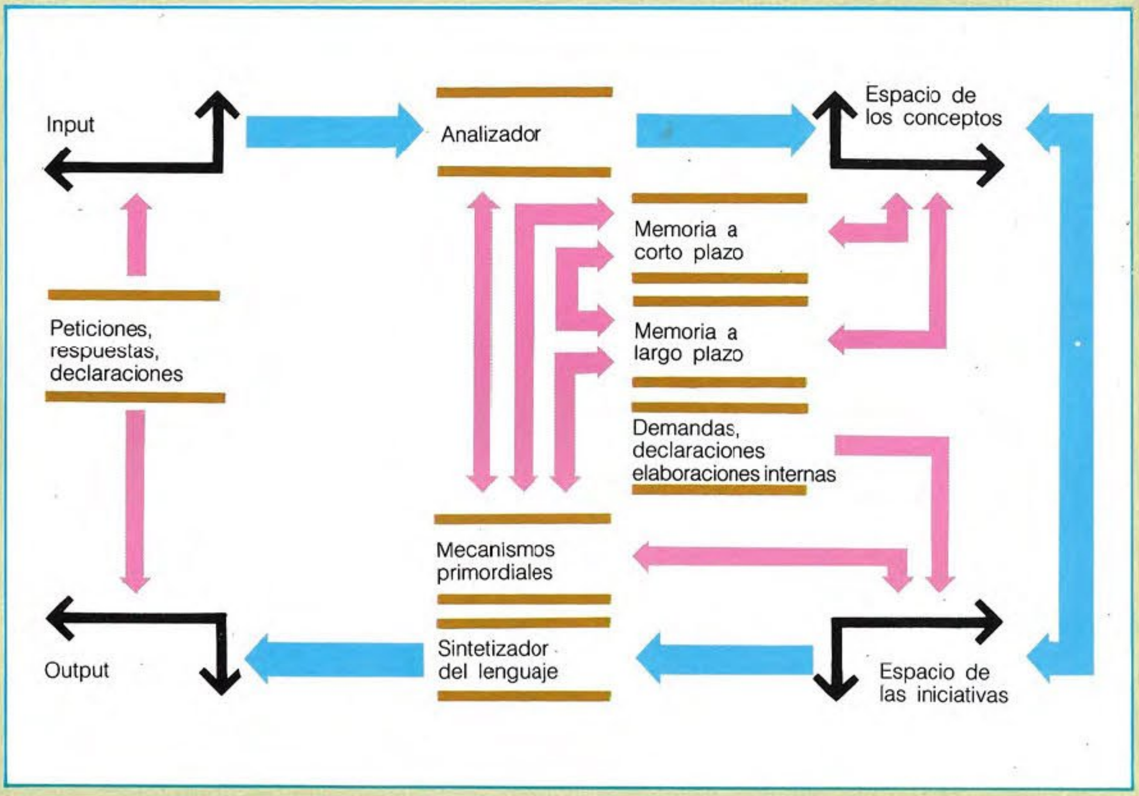
Las aplicaciones prácticas de los sistemas de reconocimiento de la voz se limitan por ahora al de palabras aisladas.

Un sistema japonés, capaz de reconocer hasta frases de tres palabras, se ha aplicado a la reserva de los billetes en la línea ferroviaria del Tokaido (el famoso tren rápido Tokyo-Osaka), mientras que en Norteamérica se está estudiando un sistema de adiestramiento de los controladores del tráfico aéreo. Un reconocedor de palabras aisladas desarrollado en el Elsag se basa en un analizador sintáctico (parser) realizado en el ámbito de la Direzione Ricerca Centralizzata. Este permite controlar con la voz un robot simulado, es decir un punto luminoso que se mueve sobre la pantalla de un tubo de rayos catódicos entre un laberinto, y obedece a una gama de comandos como: a la izquierda, a la derecha, salta una pared, etc.

También en el campo del reconocimiento de la voz es difícil preparar el pattern recognition verdadero de la lingüística computacional; esto hace la investigación aún más interesante porque cada investigación sobre la máquina se convierte también en una investigación sobre su creador: el hombre.

Para facilitar el diálogo directo hombre-procesador, es necesario que el usuario no especializado quede agilizado en sus relaciones con la máquina. Además de actuar en el sentido de permitir un diálogo con la voz, es posible facilitar todavía más la interacción hombre-máquina si el hombre se olvida de que está frente a una máquina. Debido a que esta última tiene una característica desagradable, que consiste en su repetibilidad, es decir, en el hecho de reaccionar de forma siempre idéntica ante un input dado, en Elsag se ha buscado el desarrollo de una máquina que contenga un poco de arbitrariedad típicamente humana.

Esta idea se ha concretado en el proyecto LISA



Esquema del sistema LISA (Lingüística Semántica Avanzada).

(Linguistica Semantica Avanzata). Los diversos elementos que componen el proyecto LISA son ilustrados en el esquema de arriba. Se trata en realidad de programas, es decir, de bloques de software, cada uno dotado de características apropiadas, que pueden ser conectados entre sí. Hasta ahora se han realizado dos de los bloques, el analizador y el sintetizador, y se han conectado entre sí en plan experimental, mientras que el modelo de inteligencia artificial está en fase de realización. El analizador traduce un input expresado en un subconjunto del lenguaje natural italiano, en una representación semántica que, siguiendo a R.C. Schank, se le ha dado el nombre de «conceptualización» adaptada para ser manipulada por la inteligencia artificial. Los input introducidos son frases enunciativas o interrogativas sencillas, es decir, proposiciones principales formadas a partir de los vocablos presentes en la memoria de la máquina. La característica fundamental de este sistema es el uso interconectado de criterios sintácticos y semánticos, que intervienen juntos en los diversos puntos del análisis.
Los elementos de la conceptualización o «elementos conceptuales» se han dividido en tres tipos: operadores, conceptos y modificadores. Los conceptos son todos los objetos pensables; los operadores indican acciones o relaciones entre elementos conceptuales; los modificadores sirven para especificar mejor el significado de los demás elementos conceptuales.
Esta conceptualización se analiza con un simulador de inteligencia artificial (en fase de desarrollo) que la elabora o eventualmente genera una nueva conceptualización, partiendo de las informaciones contenidas en dos memorias, una a corto plazo y otra a largo plazo. En la primera se conceptúan las últimas cinco conceptualizaciones presentadas en la entrada al simulador o generadas por éste. En la segunda, las informaciones que se tenían antes relativas al ambiente semántico elegido, que es el de las localidades geográficas italianas y el de los posibles usuarios del LISA. Una típica frase de entrada puede ser: «Luis va con frecuencia a Madrid en avión». Una vez definida la conceptualización por parte del simulador de inteligencia artificial, entra en juego el sintetizador lingüístico que la traduce en lengua italiana normal.



Experimentos con el sintetizador y el analizador del LISA.

Como se ha indicado, se ha realizado un interesante experimento, el de conectar entre sí el analizador y el sintetizador para verificar su estabilidad semántica. Esto se efectúa del siguiente modo: se pone como input una frase en el analizador, que la conceptualiza. Esta conceptualización, a su vez, se pone como input al sintetizador, que hace uso de ella para producir una frase que, a su vez, vuelve al input del analizador y así sucesivamente.

En su libro Computer power and human reason, *Weizenbaum cita el siguiente hecho: habiendo realizado en el MIT un programa (ELIZA) que simulaba las reacciones de un psicólogo analista ante las afirmaciones de un paciente según el método de Roger (que consiste en tomar esencialmente las afirmaciones del paciente y registrar sus reacciones ante lo que ha afirmado él mismo), al hacerlo probar a su secretaria, después de unas pocas respuestas, ésta le rogó que... saliese de la estancia, ya que ella tenía que confiar al calculador algo muy personal para que fuese escuchada (o vista) por él.*

Una reacción similar todavía no se ha verificado en ninguno de los investigadores dedicados al proyecto LISA, quizá porque el ambiente semántico elegido no tiene una carga emotiva tan fuerte como para poder hacer un programa que simule un psicoterapeuta y sus preguntas. Pero el propio Weizenbaum quedó un poco sorprendido al ver que su programa (un experimento de informática y nada más) era citado como ejemplo de terapia psicoanalítica puesta a disposición de todos gracias a los progresos de la electrónica. ¿Existirá alguna vez una verdadera inteligencia artificial? Las opiniones sobre este punto varían. Un cibernético como M. Schutzenberger lo niega rotundamente.

El futuro dará una respuesta a la pregunta sobre la inteligencia artificial y a su posible realización. Mientras tanto, las «máquinas cibernéticas», desde los robots industriales que sueldan las carrocerías de los automóviles hasta los lectores de direcciones postales que clasifican nuestra correspondencia, están destinados a asumir un creciente papel en nuestra vida.

(Extraído de: «Applicazioni dell'Elaboratore nel campo dell'intelligenza artificiale». de T. Chersi, en QUADERNI DI INFORMATICA, año VIII, n.° 1, 1981, Honeywell Information Systems Italia)

Como índice de variabilidad se asume la raíz cuadrada (con signo positivo) de la variancia. El valor hallado se indica con s, y se llama **desviación cuadrática media**.
Veamos ahora la comparación: ya que la desviación cuadrática media es de 71,56 ptas/kg sobre una media de 715 ptas/kg, la variabilidad del precio no es despreciable. Abajo aparece el listado de un programa que calcula la media y la desviación cuadrática media de cierto número de valores (MAX) introducido por consola. El programa memoriza los datos en la matriz A(100); por tanto, MAX no es mayor de 100.

PROGRAMA PARA EL CALCULO DE LA DESVIACION CUADRATICA MEDIA

```
50 DEFDBL A-H
60 DEFINT J-N
70 DEFDBL O-Z
80 DIM A(100)
90 '
100 INPUT "Introducir el número de observaciones"; MAX
110 IF MAX=0 OR MAX>100 THEN PRINT"** ERROR **":GOTO 100
120 T=0f
130 FOR I=1 TO MAX
140 PRINT "Observación N.:";I
150 INPUT "    Valor     :;A(I)
170 T=T+A(I)          ' Cálculo de la suma
180 NEXT I
190 TMEDIA=T/MAX
200 LPRINT " MEDIA = "; TMEDIA
250 ' CALCULO DE LA SUMA AL CUADRADO
260 T=0                ' T se utiliza nuevamente con otro significado
270 FOR I=1 TO MAX
280 T=T+A(I)*A(I)      ' T contiene la suma de los cuadrados de los datos
290 NEXT I
295 LPRINT " SUMA DE LOS CUADRADOS = ";T
300 S=SQR (T/MAX-(TMEDIA*TMEDIA))     ' S = Desviación cuadrática media
305 LPRINT " DESVIACION               = ";S
310 '
320 ' * CALCULO DEL NUMERO DE DATOS EN QUE ESTA COMPRENDIDO EL VALOR
330 '   EN EL INTERVALO ENTRE MEDIA (TMEDIA) Y DESVIACION CUADRATICA MEDIA (S)
340 N=0       ' CONTADOR
350 FOR I=1 TO MAX
360 IF TMEDIA <=A(I) AND A(I) <=S+TMEDIA THEN K=K+1
370 NEXT I
380 '
390 LPRINT " LOS VALORES COMPRENDIDOS ENTRE MEDIA Y DESVIACION SON : ";K
400 '
410 NP=K*100/MAX
420 '
430 LPRINT " CORRESPONDEN AL ";NP;" TANTO POR CIENTO"
440 '
450 INPUT " CONTINUA (SI/NO) ";RESP$
460 IF RESP$="SI" GOTO 100
470 END


*** LOS VALORES DE LOS DATOS USADOS PARA PRUEBA SON LOS MISMOS DEL TEXTO **


MEDIA = 715
SUMA DE LOS CUADRADOS = 3614425
DESVIACION               = 71.56409714305
LOS VALORES COMPRENDIDOS ENTRE MEDIA Y DESVIACION SON : 2
CORRESPONDEN AL 29 POR CIENTO
```

SOLUCIONES DEL TEST 15

1 / a) La función debe ser definida como cadena, ya que utiliza la CHR$, que restituye una cadena; por tanto, la forma exacta es:
DEF FNA$ = CHR$(27) + CHR$(10)

d) Es errónea ya que contiene un valor numérico, mientras que está definida como cadena; una forma exacta es la siguiente:
DEF FNA$ = "Cadena de prueba" + " = " + STR$(256)

---

2 / El programa puede ser el siguiente:

```
10 '   **PRINCIPIO
20 INPUT "CADENA, NUMERO ";A$,N
30 IF N < 0 OR N > 99 GOTO 20    '   CONTROL
40 IF A$="FIN" GOTO 190
50 ' ** CONTROLA SI N PUEDE SER UNA LETRA EN ASCII
60 '   SOLO MAYUSCULAS
70 K=0
80 IF 65 < N AND N < 90  THEN K = 1
90 IF K = 0 GOTO 20    ' EL NUMERO NO ES UN CARACTER
100'
110 ' ** A ESTE PUNTO SOLO SE LLEGA SI N REPRESENTA UN CARACTER
120 B$=CHR$ (N)
130 '
140 M=INSTR (A$, B$)
150 IF M=0 GOTO 20
160 PRINT "EL NUMERO "; N,  "REPRESENTA EL CARACTER "; B$
170 PRINT "Y SE ENCUENTRA EN POSICION"; M, "DE LA CADENA "; A$
180 GOTO 20
190 END
```

---

3 / El programa debe utilizar matrices y DATA. Una posible forma (téngase en cuenta que antes del programa se ha introducido la línea OPTION BASE 1) es la siguiente:

```
10 '   ** PRINCIPIO
20 DIM CODIGOS (11), NMT (10) NOMBRES$ (10)
30 FOR I = 1 TO 10 : READ NOMBRES$ (I) : NEXTI
40 DATA "PRIMER NOMBRE"
50 '
60 '                         NOMBRES PREVISTOS
70 '
130 DATA "DECIMO NOMBRE"
140 '   LECTURA DE DATOS
150 INPUT "CODIGO "; CD
155 IF CD = 0: GOTO 310   ' A LA IMPRESORA Y FIN DE PROGRAMA
160 IF CD < 10 OR CD > 20 GOTO 150
170 INPUT "IMPORTE " ; IM
180 IF IM < 1250 OR IM > 5700 GOTO 170
190 INPUT "CANTIDAD ";QT
200 INPUT "NOMBRE "; NM$
210 K = 0
220 FOR I = 1  TO 10
230 IF NM$ = NOMBRES$ (I) THEN K = I
240 NEXT I
250 IF K = 0 GOTO 200     '  K = 0 SIGNIFICA NOMBRE NO ENCONTRADO
260 TOT = QT * IM         ' IMPORTE TOTAL
270 L = CD - 9
```

```
271 ' EL CODIGO (CD) TIENE UN VALOR ENTRE 10 Y 20. LA DIFERENCIA
272 ' CD-9 VARIA ENTRE 1 Y 11 (10-9 = 1, 20-9= 11)POR TANTO INDICA A QUE
273 ' POSICION DE LA MATRIZ (CODIGOS) CORRESPONDE UN DETERMINADO VALOR
274 ' DE (CD). POR EJEMPLO SE HA INTRODUCIDO CD = 12 EL INDICE (L) VALE
275 ' 12-9 = 3 POR TANTO EL IMPORTE ASOCIADO AL CODIGO (CD) DEBE SER
276 ' MEMORIZADO EN LA POSICION 3 DE LA MATRIZ (CODIGOS)
280 CODIGOS(L) = CODIGOS (L) + TOT    ' ACUMULA EL TOTAL POR CODIGOS
290 NMT (K)=NMT (K) + TOT             ' ACUMULA EL TOTAL POR NOMBRE
291 ' SE RECUERDA QUE (K) APUNTA AL NOMBRE (INSTRUCCIONES 210-240)
300 GOTO 150                  ' INTRODUCCION DE UN NUEVO GRUPO DE DATOS
310 '   ** IMPRESION **
320 FOR I=1  TO 11                    ' SELECCIONA UNO DE LOS 11 TOTALES
330 N=I + 9                         ' RECONSTRUYE EL CODIGO
340 PRINT   "CODIGO:  ";N, "TOTAL:  "; CODIGO (I)
350 NEXT I
360 '
370 ' ** IMPRESION POR RELACION NOMINATIVA **
380 FOR I=1 TO 10          ' SELECCIONA UNO DE LOS 10 NOMBRES
390 PRINT   "NOMBRE :  "NOMBRES (I),  "TOTAL :  ";NMT (I)
400 NEXT I
410 END
```

4 / El importe unitario medio se obtiene sumando todos los importes leídos en la línea 170 y dividiéndolo por el número de entradas realizadas. La subrutina debe contener un contador que se incrementa con cada entrada y un totalizador en el que se acumulan todos los importes. Al final del programa, su cociente proporciona el valor medio. Esta rutina puede llamarse insertando en el programa anterior la línea:

```
185 GOSUB 1000
```

y debe contener sólo dos instrucciones:

```
1000' *Entrada
1010 CUENTA = CUENTA + 1'contador
1020 SUMA = SUMA + IM'totalizador
```

Al final del programa, antes de la línea 410, debe insertarse el cálculo del valor medio buscado: SUMA/CUENTA.

5 / El test se resuelve con un bucle entre 1 y NUM, con NUM = Valor transmitido a la subrutina, del que se quiere calcular el factorial.

```
1000 'cálculo del factorial del número NUM
1010 T = 1 'Primer valor
1020 FOR I = 2 TO NUM 'Parte de 2, pues el valor 1 está asignado a la línea 1010
1030 T = T * I
1040 NEXT I
1050 RETURN
```

En la salida de la variable T se tiene el factor de NUM.

6 / Este punto se resuelve utilizando la subrutina anterior para el cálculo de los tres factores [N!, K!, (N – K)!] que aparecen en la fórmula

```
10 '** Principio
15 DEFDBL R – T
```

```
 20 INPUT "Número de elementos"; N
 30 INPUT "Clase (número de elementos por grupo)"; K
 40 NUM = N
 50 GOSUB 1000
 60 S = T    'Memoriza en S la respuesta de la subrutina (factorial de N)
 70 NUM = K
 80 GOSUB 1000
 90 S1 = T    'En S1 hay el factorial de K
100 NUM = N - K
110 GOSUB 1000
120 S2 = T    'S2 = (N - K)!
130 R1 = S/S2:R = R1/S1    'R es el número buscado
140 PRINT "Las combinaciones son"; R
150 END
```

Obsérvese que la línea 130 resuelve el cálculo final. La forma más inmediata es R = S/(S1 * S2), pero a causa de los órdenes de magnitud de los números S1 y S2, su producto podría generar un valor que sobrepasara la precisión de la máquina. Desarrollando primero la división S/S2 se reduce la posibilidad de error. Además, con un programa estructurado así, el máximo número N que puede tratarse es aproximadamente de 30 (para valores superiores se tienen errores de precisión).

Las funciones de entrada y de salida de datos

Se exponen aquí los comandos y las funciones para la introducción de los datos por consola y para su envío a la pantalla o a la impresora. Los comandos y las funciones que llegan a la unidad de disco y de cinta, a pesar de expresar funciones I/O, se tratan por separado. En los lenguajes de alto nivel, los datos introducidos o enviados a la impresora, antes se elaboran con rutinas de sistema. En la introducción, estas últimas convierten el formato generado por el teclado en el requerido por la máquina y en el envío adaptan el formato de los caracteres a las necesidades de los periféricos de salida.

Las modalidades de la conversión dependen del tipo de dato introducido: si éste es un número entero, las rutinas de sistema lo convierten en binario y lo depositan en dos bytes de memoria contiguos. Los números reales en el interior de la máquina son representados en forma exponencial (coma flotante) y la rutina de entrada debe proceder a separar la mantisa del exponente. El número 65743 podría memorizarse como 65.743E3 (E indica el exponente). La posición del punto decimal y el número de cifras que pueden memorizarse dependen del tipo de máquina y del sistema operativo.

La rutina de entrada de datos, además de la función de «formateado», también realiza todos los controles necesarios para asegurar la fidelidad formal de los datos introducidos. En particular tiene la misión de examinar si existe congruencia entre el tipo de dato proporcionado y el solicitado, y en el caso de introducción de más valores, si la cantidad de los datos introducidos es igual a la que se espera. Los datos de impresión (o de escritura en pantalla) son tomados de la memoria y modificados tal como lo pide el programa de aplicación y después enviados a los periféricos. La operación de modificación formal de los datos para adaptarlos a las necesidades particulares de impresión se llama **formateado** (del vocablo inglés "format" que, en muchos lenguajes de programación, coincide con el código de una instrucción que indica cómo debe ser realizada la impresión). Los controles a realizar, los formatos y la modalidad de adquisición o los formatos de impresión de datos pueden ser controlados por el programador con las adecuadas instrucciones y funciones.

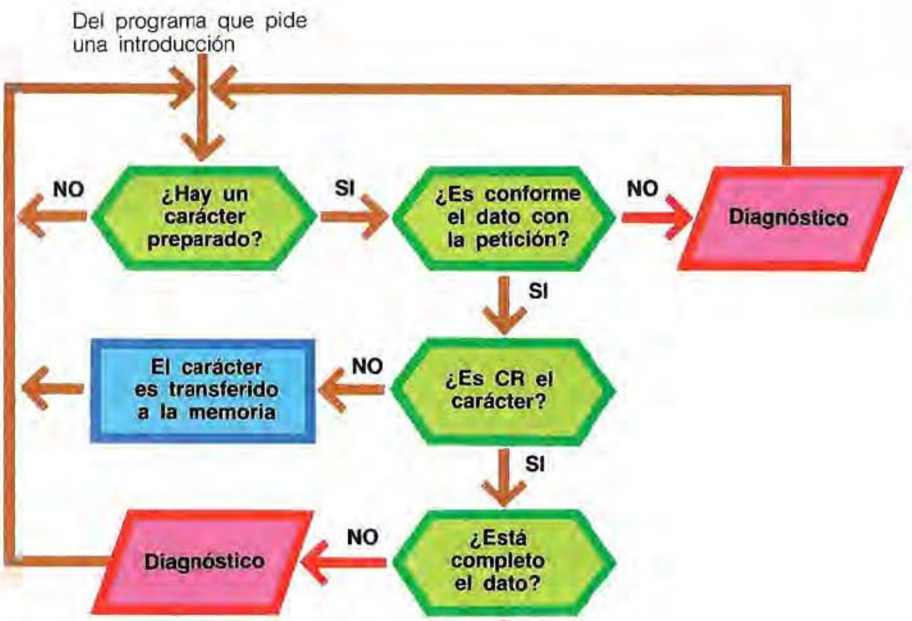
Funciones de introducción de datos

Cada fase de entrada de datos termina pulsando la tecla CR (o RETURN). Si los datos introducidos son correctos, se transmiten al programa de aplicación; si se encuentran errores, aparece un mensaje de diagnóstico. En Basic, algunos errores en la base de introducción pueden recuperarse; en otros lenguajes se puede interrumpir el programa. Abajo se ha indicado un esquema de principio de las funciones desarrolladas en la fase de adquisición de datos.

Cuando encuentra una instrucción de introducción, el programa de aplicación llama a la correspondiente rutina de sistema. La primera función desarrollada por esta rutina es la espera de un carácter. El ordenador emplea un tiempo brevísimo para abrir el canal de introducción. El sistema entra así en un bucle, cuyo único objeto es el de esperar la llegada de un caracter. Cuando la introducción termina, se realiza un primer control formal; por ejemplo, si se pide un valor numérico, el sistema controla que no se trata de una letra. Si el carácter se acepta, el ordenador se pone nuevamente en el bucle de espera. Los siguientes caracteres se colocan el uno al lado del otro a medida que van llegando, de manera que pueda formarse el dato completo. La colocación termina cuando el operador pulsa la tecla CR (o RETURN). Si todos los datos pedidos están presentes el ordenador vuelve al programa de aplicación; si no es así, se pone en espera de los otros datos necesarios para completar la petición.

INPUT. La principal instrucción de introducción es la siguiente: INPUT "Mensaje"; VARIABLES. La cadena "Mensaje" es cualquier palabra o frase que se desea presentar en pantalla en el momento de la introducción. El sistema escribe en pantalla todo lo que hay encerrado entre las comillas y pone en evidencia al final del mensaje un signo de interrogación, para indicar que está en espera. El operador, entonces puede introducir los datos, que son asignados secuencialmente a las variables especificadas en la instrucción. Por ejemplo, la instrucción

ESQUEMA DE PRINCIPIO DEL MECANISMO DE ADQUISICION DE DATOS



INPUT "Prueba"; V1,K$,B

produce la aparición en pantalla del mensaje

¿Prueba?

El operador deberá introducir tres datos (V1,K$,B), dos de los cuales (el primero y el tercero) serán valores numéricos, el otro será una cadena. Cada dato deberá estar separado por una coma y al final de la introducción el operador deberá pulsar la tecla CR.
La introducción 3.4,ABC,100 satisface la instrucción anterior. Invirtiendo el orden de los datos, por ejemplo 3.4,100,ABC, no se tiene congruencia con el formato deseado (número, cadena, número) y el programa indica error.
En la fase de introducción puede suprimirse el interrogante (emitido automáticamente por el sistema) sustituyendo el símbolo ; por ,.
En la forma

INPUT "Prueba", V1,K$,B

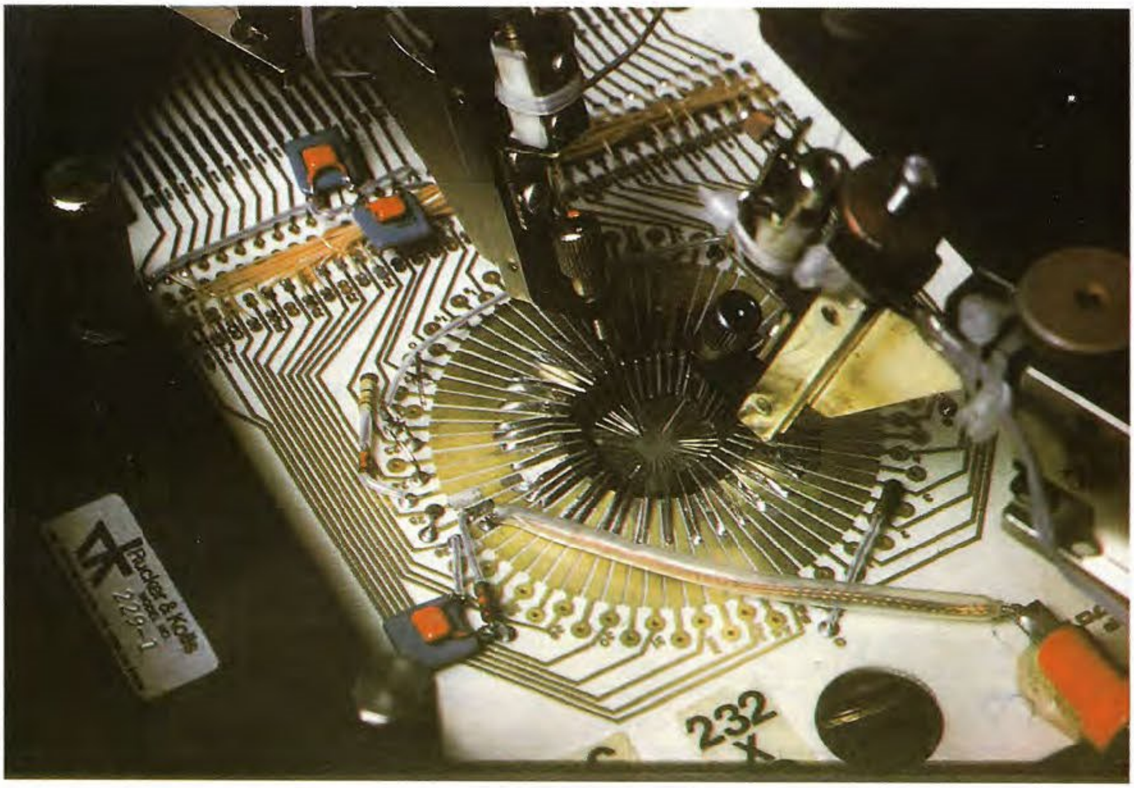
la instrucción considerada anteriormente conserva el mismo efecto con la eliminación del símbolo ? después de la descripción.
La última opción aceptada de la instrucción INPUT permite insertar el símbolo ; seguido después del código:

INPUT; "Descripción"; V1,K$,B

Esta forma suprime el retorno del cursor al principio cuando el operador pulsa la tecla CR para terminar la introducción.
En condiciones normales, el código CR (carriage return) indica a la máquina el término de la introducción y, al mismo tiempo, posiciona el cursor al principio de la siguiente línea. Con esta última opción, y utilizando la instrucción INPUT, es posible inhibir el movimiento del cursor para tener más entradas en la misma línea.
El uso de la instrucción INPUT, en una cualquiera de sus formas, implica tres limitaciones:

1 / los caracteres introducidos son escritos automáticamente en la pantalla
2 / la longitud de cada dato (el número de cifras para los datos numéricos o de caracteres para las cadenas) debe ser gestionada por el operador
3 / no se pueden introducir caracteres especiales

Durante la introducción de los datos, cada carácter se introduce directamente al programa y, al mismo tiempo, su eco es presentado en la pantalla. En algunos casos, por ejemplo en la fase de introducción de las palabras de orden en programas reservados, este tipo de funcionamiento debe ser evitado, y debe utilizarse una instrucción de entrada que no provoque el eco en la pantalla.
El segundo punto (control de la longitud del dato) es muy importante para evitar errores de introducción y está en la base de la gestión de la máscara de vídeo. En este tipo particular de gestión de datos, el usuario tiene a su disposición un campo bien definido en el que puede introducir los valores de input, valores que sin embargo no deben superar la longitud prevista.
Con el uso de la instrucción INPUT, el programa no está bajo control: mientras los datos se introducen no es posible efectuar ninguna verificación. Pero controlando la longitud del dato a medida que se digita, se puede interrumpir la introducción en el momento oportuno.
También el último punto está ligado a empleos específicos. En la fase de adquisición con la instrucción INPUT, el Basic controla que los valores introducidos sean congruentes con las variables que se le asignan y descarta eventualmente códigos particulares que, en algunas aplicaciones, pueden ser útiles para comunicar al ordenador instrucciones en lugar de datos. También este funcionamiento es característico de las máscaras vídeo. Un máscara puede proporcionar numerosos campos de introducción y, en el momento en que se presenta en pantalla, espera efectivamente la introducción de todos los datos requeridos. En caso de que no se hubiese previsto el uso de códigos que puedan indicar el término de las operaciones de introducción, el usuario quedaría obligado a completar siempre todos los campos de la máscara, con notable pérdida de tiempo.
En cambio, si se prevé la posibilidad de interrumpir la fase de introducción, pueden proporcionarse cada vez únicamente los datos que son necesarios en aquel momento. La instrucción INPUT A,B,C$,D necesita en cualquier caso cuatro valores (dos numéricos, una cadena, un número) y no es posible introducir una cantidad menor de datos. El operador debe proporcionar los diversos valores de modo que resulten correspondientes a lo especificado en la misma instrucción de introducción.
En particular, para la cadena C$, es necesario introducir por lo menos un carácter (los espa-



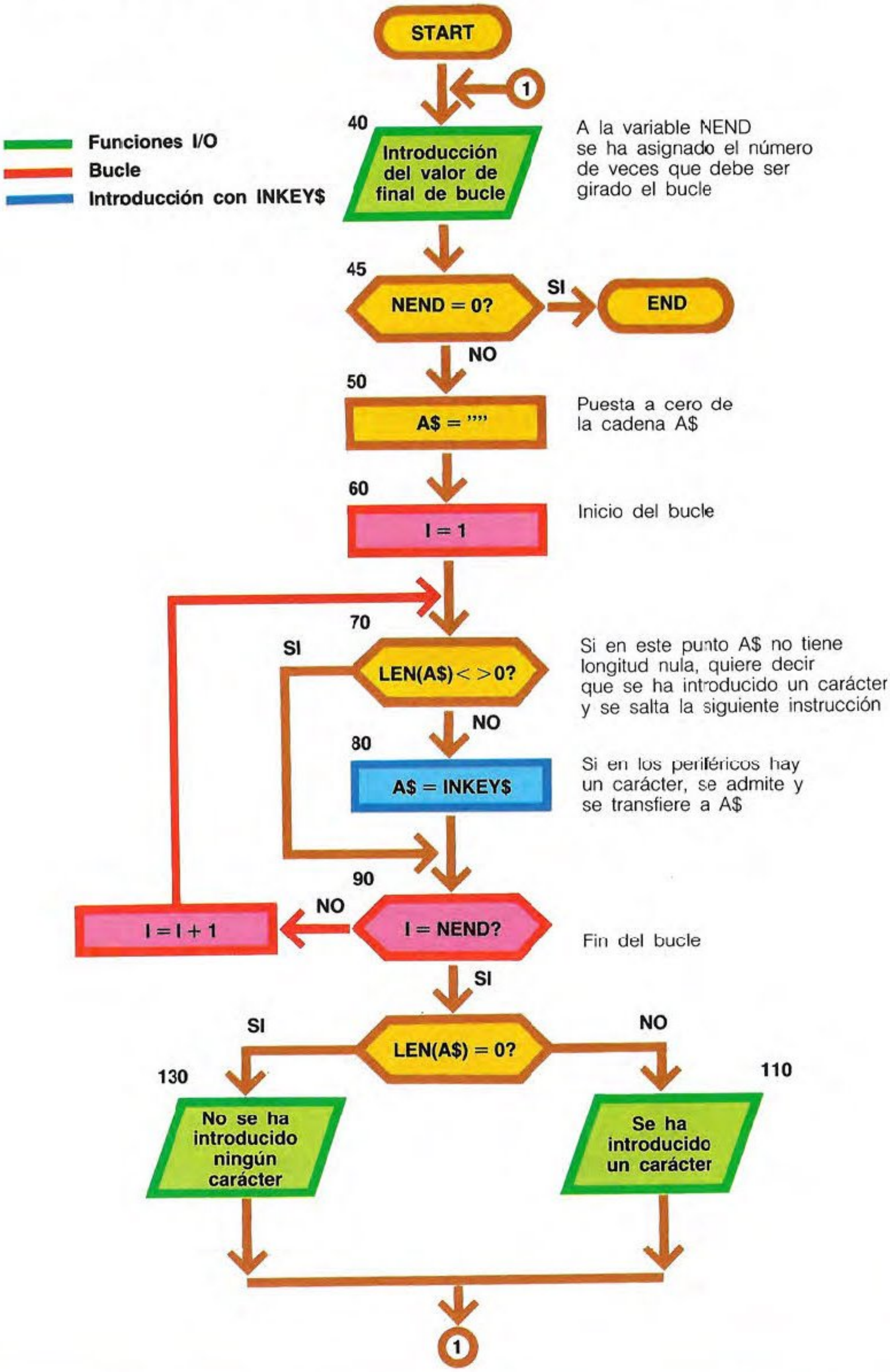
K. Reese/Marka

La fase de control computerizado de un microcircuito sobre chip.

cios en blanco solos no son aceptados).
Como puede verse, de este modo no es posible introducir por teclado una cadena nula o una cadena que sólo contenga espacios.
Estas dificultades desaparecen con el uso de las instrucciones INKEY$ INPUT$(N), aunque se crea una nueva complicación, ya que todos los controles (por ejemplo de la numericidad del dato) que antes eran realizados por las rutinas del sistema, ya no son activos y, por tanto, deben ser asumidos por el programa de aplicación. La elección de una u otra forma depende del tipo de aplicación. En un programa sencillo, basta el uso de la instrucción INPUT. Si no es así, resulta indispensable el uso de las formas alternativas de adquisición de datos.

**INKEY$**. Permite la adquisición de un solo carácter por teclado; el carácter introducido no tiene eco y, por tanto, no aparece en la pantalla. Con la INKEY$ se adquiere cualquier código, comprendidos los caracteres de control, a excepción del código CNT + C (tecla CONTROL y letra C pulsadas al mismo tiempo) que produce la conclusión del programa (esto no sucede

EJEMPLO DE FUNCIONAMIENTO DE LA INSTRUCCION INKEY$



EJEMPLO DE USO DE LA FUNCION INKEY$

```
10 '
20 ' *** EJEMPLO DE USO DE LA : INKEY$
30 '    FILE : INKEY
40 INPUT "Número de veces que debo desarrollar el BUCLE "; NEND
45 IF NEND=0 GOTO 150
50 A$=""
60 FOR I=1 TO NEND
70 IF LEN(A$) <>0 GOTO 90
80 A$=INKEY$
90 NEXT I
100 IF LEN(A$)=0 GOTO 130
110 PRINT " Se ha introducido un carácter durante el BUCLE"
120 GOTO 40
130 PRINT " NO SE HA INTRODUCIDO NINGUN CARACTER"
140 GOTO 40
150 END
```

en el Basic compilado, en el que también el código CNT + C se convierte en transparente y no se introduce en el programa de aplicación). La sintaxis de la instrucción es

A$ = INKEY$

En la cadena A$ se transfiere el carácter leído por el terminal (A$ es una cadena cualquiera definida por el usuario).

Esta función de entrada no espera el dato: el programa continúa girando y al mismo tiempo se controlan los periféricos; cuando el carácter está preparado viene la adquisición y se asigna a la cadena especificada en la instrucción.

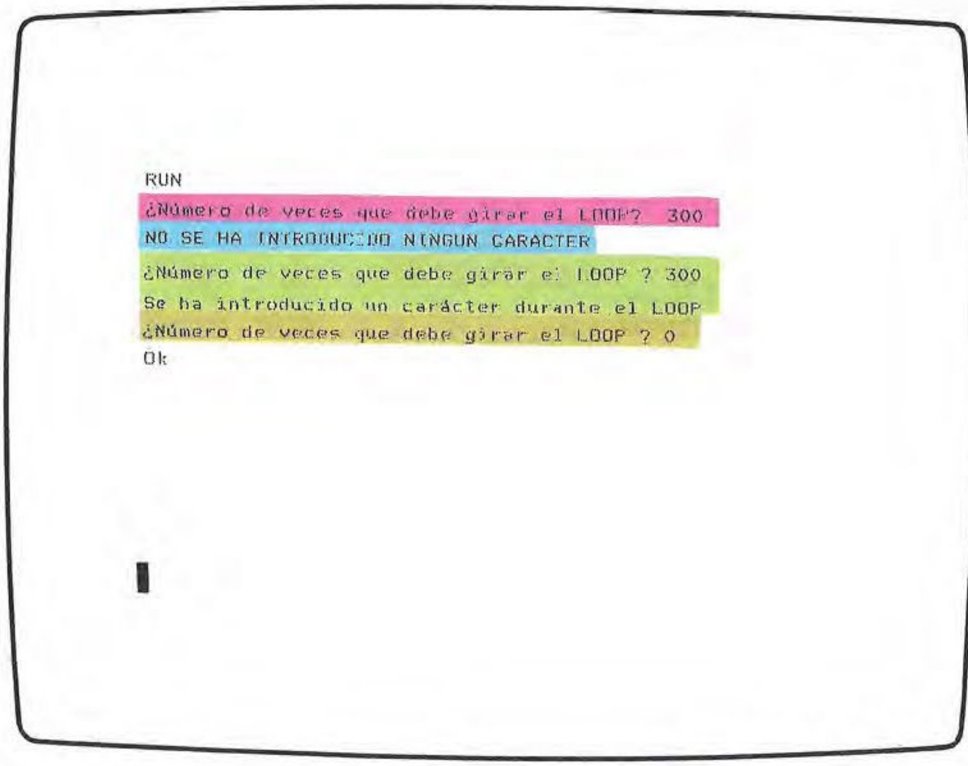
En la página 551 y encima se indican el diagrama de flujo y el listado de un programa ilustrativo que utiliza la instrucción INKEY$. Este programa es sustancialmente un bucle que contiene la instrucción INKEY$. En cada giro del bucle, la instrucción controla el estado de la unidad de entrada (teclado) y si encuentra un carácter preparado (pending) lo transfiere a la cadena A$. A partir de este momento, la variable A$ ya no tiene una longitud nula (contiene el carácter introducido): la condición de la instrucción 70 se convierte en verdadera [LEN(A$)<>0] y a continuación la instrucción INKEY$ se salta. Al final del bucle, la variable A$ contendrá un carácter, o bien todavía será una cadena nula si el operador no ha efectuado ninguna introducción. Controlando la longitud (instrucción 100) se pueden distinguir los dos casos y seleccionar el mensaje correspondiente.

EJEMPLO DE USO DE LA FUNCION INKEY$

La función de input INKEY$ permite la adquisición de un carácter de los periféricos (consola-vídeo).
En esta página se muestra el output del programa de ejemplo indicado en la página anterior.



Después de la activación de la ejecución (comando RUN), el control pasa a la función de input que se llama en la línea 40.



La fase de interpretación de esta línea produce la visualización del mensaje que el programador ha incluido entre comillas.
El signo de interrogación que sigue lo pone automáticamente el sistema, mientras que el número 300 ha sido digitado por el operador.
La función INPUT asigna el valor 300 a la variable NEND, límite superior del índice del bucle.

Apenas el operador pulsa la tecla RETURN después de la digitación del número 300, el control pasa sucesivamente a las líneas 45 (control de fin de RUN), 50 (puesta a 0 de la cadena A$) y entra en el bucle (línea 60), que contiene la función INKEY$.

Durante el desarrollo del bucle, el control no ha encontrado ningún carácter «pendiente» en la puerta de entrada; la condición de la línea 100 ha resultado verdadera y se ha ejecutado la instrucción de impresión que está presente en la línea 130.

Esta fase es la repetición de la anterior, pero esta vez el operador ha digitado un carácter cualquiera durante la ejecución del bucle. El carácter no aparece en pantalla porque la función INKEY$ suprime el eco. La condición en la línea 100 resulta ahora falsa y el control pasa a la línea 110, que produce la visualización del mensaje.

Para terminar, el operador introduce el código 0. El control previsto en la línea 45 determina el salto al final del programa.

La instrucción INKEY$ tiene un funcionamiento muy diferente al de las otras instrucciones de entrada. Si se sustituye la fila 80 (A$ = INKEY$) con cualquier otra instrucción de entrada, el programa se detiene en esta instrucción esperando el dato. Empleando la instrucción INKEY$ no se produce la espera: el sistema continúa con la actividad principal teniendo bajo control la puerta de entrada y admite el dato apenas está preparado. Abajo se esquematizan las lógicas de los dos tipos de introducción.

INPUT$(N). Adquiere N caracteres por teclado sin producir el eco (no aparecen en pantalla). Por ejemplo, A$ = INPUT$(6) adquiere 6 caracteres y los transfiere a la cadena A$.

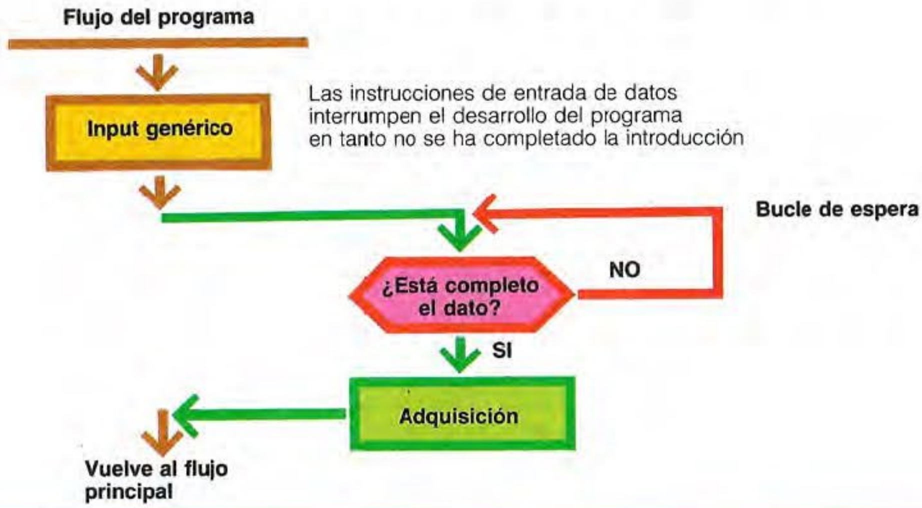
En este caso, la introducción de los datos no debe terminarse con la tecla CR, ya que el sistema tiene en cuenta el número de caracteres deseados y los introducidos; cuando ambos números son iguales, abandona la instrucción de entrada y vuelve al flujo principal.

La función INPUT$(N) (no es una instrucción) no

ESQUEMA DE FUNCIONAMIENTO DE LA INSTRUCCION INKEY$



ESQUEMA DE FUNCIONAMIENTO DE LAS OTRAS INSTRUCCIONES DE ENTRADA DE DATOS



EJEMPLOS DE USO DE LA FUNCION INPUT$(N)

```
10 ' ** EJEMPLOS DE USO DE LA FUNCION INPUT$(N)
20      FILE   :  INPUT
30 DEFINT A-Z
40 INPUT" Cuántos caracteres se desea adquirir ";N
50 IF N<=0 OR N>80 GOTO 40        ' CONTROL SOBRE EL NUMERO N DE CARACTERES
60 '
70 INPUT "Se quiere el eco (Responder SI o NO) ", RESP$
80 '
90 ' En el input de la línea 70 se ha utilizado el formato que
100 ' suprime el interrogante emitido por el sistema
110 '
120 IF RESP$<>"SI" AND RESP$<>"NO" GOTO 70 ' CONTROL
130 PRINT "Introducir ";N;" caracteres. "
140 A$=INPUT$(N)
150 IF RESP$="SI" THEN PRINT A$
160 '
170 ' * SEGUNDO METODO
180 '
190 PRINT "Segundo método: A$ se reinscribe carácter por carácter"
200 A$=""
210 FOR I=1 TO N
220 B$=INPUT$(1)     ' LECTURA DE UN SOLO CARACTER CADA VEZ
230 PRINT B$;
240 A$=A$+B$          ' EL CARACTER LEIDO SE ACUMULA EN A$
250 NEXT I
260 '
270 INPUT " CONTINUAR (SI/NO)";RESP$
280 IF RESP$="SI" GOTO 40
290 END
```

tiene eco; por tanto, para visualizar los caracteres introducidos deben reescribirse con una instrucción adecuada contenida en el programa. Arriba se ve el listado de un programa para el uso de INPUT$(N). Se han previsto las dos formas: con y sin eco. Si se desea el eco, debe adquirirse y escribirse en la pantalla un carácter cada vez, de modo que el eco aparezca inmediatamente después de la digitación.

LINE INPUT. Permite adquirir una línea entera de caracteres (normalmente hasta un máximo de 256). Por ejemplo, la línea

LINE INPUT "Descripción"; A$

adquiere en A$ una cadena introducida por teclado. La introducción prosigue también sobre más líneas de la pantalla de vídeo: el operador puede escribir en ella más líneas, utilizando para ir al principio la tecla LF (line feed); el fin de la introducción se indica con la tecla CR.

Funciones de emisión de datos

En las funciones de emisión (impresión o escritura en pantalla), el sistema procede a la codificación de los datos y a su preparación en los formatos deseados por el usuario.

Existen varios modos de escribir los datos; los valores numéricos pueden tener un número variable de decimales, o pueden necesitar caracteres especiales. Cada uno de estos aspectos que puede asumir en la impresión un mismo dato define un formato. El usuario puede especificar con las instrucciones adecuadas el formato que desea o pedir una emisión sin formato, en cuyo caso el sistema adopta, para los datos numéricos, la forma exponencial, mientras que las cadenas se escriben sin modificarse.

El formato exponencial adoptado por omisión (es decir, por falta de indicaciones) está constituido por la mantisa (las cifras que componen dicho número, separadas por el eventual punto decimal) y por la notación E ± nn, donde E indica la multiplicación por una potencia de base 10, mientras que ± nn es el exponente de la potencia de 10 por la que se multiplica la mantisa. El eventual signo positivo del exponente puede omitirse. Por ejemplo, la notación 4931.3579E2 significa 493135.79 (4931.3579 × 100, siendo E2 = 100). Si la cantidad de cifras que componen el número no supera el máximo permitido para los reales (o para la doble precisión), el exponente Enn no se imprime, y la salida es un número con las eventuales cifras decimales separadas por un punto.

Las instrucciones de salida más sencillas, que utilizan el formato de omisión, son PRINT para la pantalla y LPRINT para la impresora. Para utilizar un formato particular, la instrucción es PRINT USING... (literalmente USANDO IMPRESION...) para la pantalla y la análoga LPRINT USING... para la impresora. A continuación, las instrucciones se refieren sólo a la impresora (LPRINT); las análogas para la unidad de vídeo se obtienen sustituyendo el código LPRINT por PRINT. Como la pantalla suele tener 80 columnas útiles y la impresora llega a tener 132 y más, pueden crearse problemas de longitud de línea, aunque superables repartiendo en dos líneas la escritura en pantalla.

LPRINT. La forma general de la instrucción es:

LPRINT "Comentarios"; VARIABLE1; ...

El Basic imprime todo lo comprendido entre los símbolos " y los valores de la variable especificada en la instrucción (VARIABLE1, etc.). Por ejemplo, las líneas

10 V1 = 3.516
20 V2 = 75
30 LPRINT "Salida" = ",V1,V2

produciendo la impresión de la línea

Salida = 3.516   75

EJEMPLOS DE USO DE LA INSTRUCCION "LPRINT"

```
10 ' *** EJEMPLOS DE USO DE LA INSTRUCCION (LPRINT) ***
20 '                        FILE = LPR
30 A1=123
40 A2=456
50 A3=789
60 LPRINT "PRUEBA N.1", A1, A2, A3
65 LPRINT                                ' ESPACIADO
70 LPRINT "PRUEBA N.2", A1; A2; A3
75 LPRINT                                ' ESPACIADO
80 LPRINT "PRUEBA N.3"; A1; A2; A3
85 LPRINT                                ' ESPACIADO
90 END
```

```
PRUEBA N.1       123          456          789

PRUEBA N.2       123 456 789

PRUEBA N.3 123 456 789
```

```
10 ' *** USO DE LA INSTRUCCION (LPRINT) CON LAS CADENAS ***
20 '                        FILE = LPRS
30 A$="MADRID"
40 B$="ES"
50 C$="GRANDE"
60 LPRINT "PRUEBA N.1";A$;B$;C$
70 LPRINT
80 LPRINT "*** ATENCION ESTA PREVISTO EL CORRECTO ESPACIADO "
90 LPRINT "     EN FASE DE ESCRITURA ***"
100 LPRINT
110 LPRINT "PRUEBA N.2",A$,B$,C$
120 END
```

```
PRUEBA N.1MADRIDESGRANDE

*** ATENCION: ESTA PREVISTO EL CORRECTO ESPACIADO
     EN FASE DE ESCRITURA ***

PRUEBA N.2       MADRID        ES          GRANDE
```

Los valores numéricos se escriben con un factor de escala (formato exponencial Enn) si superan el número de cifras máximo permitido por el tipo adoptado (6 cifras para los reales, 15 para la doble precisión), de otro modo, el factor de escala se omite. En la impresión, el Basic divide la línea en campos de 14 caracteres y permite la utilización de estos espaciados por medio de la simbología adoptada en la instrucción. Separando las variables con una coma, se tiene la impresión de cada variable al principio de la propia zona; en cambio, utilizando como símbolo de separación el punto y coma, se tiene la impresión de cada variable inmediatamente después de la anterior. Estas dos simbologías también pueden utilizarse al cerrar la instrucción; en tal caso, la siguiente instrucción LPRINT prosigue la impresión en la línea anterior sin ir de nuevo al principio. En otras palabras, es como si la nueva LPRINT fuese una continuación de la línea anterior. En la página de la izquierda se indican algunos ejemplos de impresión de variables numéricas y de cadenas.

TAB(N). En las funciones de emisión de datos, y principalmente en la impresión de tabulados, a menudo es necesario posicionar los campos de impresión con una libertad mayor que la permitida con la instrucción LPRINT (o PRINT).
Utilizando la opción ; o la opción , es posible obtener la impresión de los datos de modo seguido o distribuidos en zonas de 14 caracteres. Para obtener espaciados diferentes se utiliza la función TAB(N). Esta función separa N caracteres la zona de impresión del dato; por ejemplo, la instrucción LPRINT TAB(20); B$ produce la impresión de la variable de la cadena B$ a partir de la posición 20.
Más adelante se presentarán algunos ejemplos de empleo de la función TAB(N), cuando se hable de la estructuración de la impresión.

LPRINT USING. Permite la impresión de datos numéricos o cadenas de acuerdo con un formato definido por el usuario. La sintaxis es:

LPRINT USING "Especificación formato";
VARIABLES

Las especificaciones de formato definen, con simbologías convencionales, algunos modos particulares de emisión de los datos. Los formatos del Basic 80 se indican a continuación.

- Variables de cadena

"!"

Sólo imprime el primer carácter de la cadena

"\ \"

Imprime un número de caracteres iguales a los espacios en blanco contenidos entre los símbolos \ más 2.
Por ejemplo, las líneas:
10 A$ = "PRUEBA DE IMPRESION"
20 LPRINT USING "\ \"; A$
30 ' (2 espacios entre los símbolos \)
producen la impresión de 4 caracteres de la cadena A$ (2 espacios + 2)

"&"

Imprime la cadena sin ningún control de longitud

- Variables numéricas

"# # . #"

Especifica el número de cifras a imprimir distinguiendo entre la parte entera y los decimales.
El número de cifras de la parte entera es igual al número de símbolos que preceden al punto.
Por ejemplo, con las instrucciones
10 R = 35.716
20 LPRINT USING "# #. #"; R
se ha impreso 35.7 (2 enteros y un decimal)

"+"

Permite escribir el signo del dato antes o después de su valor numérico. En algunas aplicaciones comerciales, lo normal es escribir el signo al final del número; así, el valor – 519 se imprime como 519 –. Por ejemplo, las líneas
10 R = – 125.365
20 PRINT USING "+ # # # . # #"; R
30 PRINT USING "# # # . # # +"; R
producen las dos salidas
– 125.36 (línea 20: se necesitan sólo 2 decimales)
125.36 – (línea 30)

"–"

Sólo puede ser posicionado al término de la cadena que expresa el formato ("# # . # –") y produce la impresión del eventual signo al final del número; es equivalente a la opción "+" colocada al final del campo, con la única diferencia de que el signo sólo se imprime si es negativo

"∗∗"

El doble asterisco produce el rellenado de eventuales espacios no ocupados por el dato con el símbolo ∗. Utilizando el formato "#", la parte entera del número a imprimir puede contener un número de cifras menor del que se dispone en el campo de impresión. Si no se especifican otras opciones, en el campo de impresión se dejarán tantos espacios en blanco como cifras falten. Utilizando la opción "∗ ∗", los puestos que faltan son rellenados con el símbolo ∗ (uno por cada cifra que falta).
Por ejemplo, imprimiendo el número 7.1543, el formato "# # # # . # #" da como salida 7.15, mientras que el formato "∗ ∗ # # # # . # #" da ∗ ∗ ∗ 7.15

"$$"

Produce la impresión del símbolo $ inmediatamente antes del número. Si se emplea este formato, el eventual signo negativo debe colocarse necesariamente al final del número. Por ejemplo, el formato "$$ – # # . #" es erróneo (signo – antes del número); la forma exacta es "$$ # # . # –"

"∗∗$"

Combina los efectos de los dos anteriores. Rellena los espacios en blanco con el símbolo ∗ y pone el símbolo de dólar ($) al principio del número

","

La coma puede utilizarse para indicar los miles según la simbología anglosajona y se posiciona en cada tres cifras de la parte entera a partir de la derecha. En los formatos debe escribirse siempre a la izquierda del punto decimal. Por ejemplo, el formato "# # # #,.# #" aplicado al número 7563.121 produce la salida 7,563.12
La única variante permitida a este formato consiste en colocar la coma al final del mismo (# # # # . # #,). En este caso, el símbolo coma se imprimirá al terminar el número. La impresión del número anterior se convierte en 7563.12,

"XXXX"

Cuatro caracteres cualesquiera comprendidos en un formato especifican la notación exponencial. Los cuatro caracteres son ficticios y tienen el único objeto de reservar el espacio necesario para escribir la notación exponencial E + nn (con nn = exponente), que precisamente está constituida por cuatro caracteres. El formato "# # . # AAAA", aplicado al número 1200 produce la impresión 12.0E + 02 (E + 02 = 10^2 = 100; 12.0 × 100 = 1200) puesto que se necesitan 2 enteros y un decimal (# # . #) con formato exponencial (AAAA)

"%"

Si está incluido en un formato, se imprime al principio del dato si éste supera la longitud del campo declarado. Así, 1470.65 impreso con el formato "% # # . #" se convierte en % 1470.6, porque el número de cifras enteras supera el previsto en el formato.

En el listado de la pág. 559 se han incluido ejemplos de uso de los diversos formatos*.
Debe tenerse presente que en la fase de impresión se tiene el redondeo automático de los datos numéricos cuya parte decimal supera el campo declarado; así, el número 412.67 impreso con el formato "# # # . #" se convierte en 412.7, ya que sólo se pide una cifra decimal.
El redondeo se realiza en base a las reglas de cálculo normales: si el valor de la cifra que debe despreciarse es mayor de 5, se añade 1 a la

* Algunas impresoras indican el símbolo # con £.

EJEMPLOS DE USO DE LA INSTRUCCION "LPRINT USING"

```
10 ' ** EJEMPLOS DE USO DE LA INSTRUCCION (LPRINT USING)
20 '
30 '                  FILE = LPRUSI
40 ' *** OPCION "!"  (IMPRIME SOLO EL PRIMER CARACTER DE LA CADENA)
50 A$="MADRID DE NOCHE"
60 LPRINT USING "!"; A$
70 LPRINT                    ' ESPACIADO
80 '
90 ' *** OPCION "£££.£"    (VARIABLES NUMERICAS)
100 R=123.789       ' N. EL NUMERO TIENE TRES ENTEROS Y TRES DECIMALES
110 LPRINT USING "£££.£"; R     ' CONTINUAR LA IMPRESION DE (R) CON TRES
120 '                            ENTEROS Y UN SOLO DECIMAL
130 LPRINT                      ' ESPACIADO
140 '
150 ' *** OPCION " % "
160 LPRINT USING " % ££.££"; R  ' EL SIMBOLO % SIRVE PARA EVIDENCIAR SI EL
170 ' NUMERO IMPRESO SALE FUERA DEL FORMATO DE IMPRESION.
180 LPRINT USING " %£££.££";R
190 LPRINT: LPRINT              ' ESPACIADO
200 '
210 ' ***OPCION "+£££.££"
220 LPRINT USING "+£££.££"; R  'IMPRIME (R) CON TRES ENTEROS, DOS DECIMALES
230 '                           Y CON EL SIGNO ALGEBRAICO A LA IZQUIERDA.
240 ' *** OPCION "£££.££+"
250 LPRINT USING "£££.££+"; R '  COMO ANTES, PERO CON EL SIGNO A LA DERECHA.
260 LPRINT
270 LPRINT USING "+£££.£££"; R ' IMPRIME (R) CON UN FORMATO QUE LO COM=
280 '                           PRENDA TODO.
290 LPRINT USING "£££.£££+"; R ' COMO ANTES, PERO CON EL SIGNO A LA DERECHA.
300 LPRINT
310 '
320 ' *** OPCION "£££.£££-"   ' N. ESTE FORMATO PONE EL SIGNO ALGEBRAICO
330 '  A LA DERECHA, PERO OMITIENDOLO EN CASO DE VALOR POSITIVO.
340 Q=-456.789       ' (VALOR NEGATIVO)
350 LPRINT USING "£££.£££-"; R
360 LPRINT USING "£££.£££-"; Q
370 END
```

```
R

123.8

% % 123.79
% 123.79


+123.79
123.79+

+123.789
123.789+

123.789
456.789-
```

anterior, de otro modo (si es menor o igual a 5) se cortan sin ningún ajuste. Por ejemplo, utilizando un solo decimal, el número 56.86 se convierte en 56.9 (la cifra que debe eliminarse vale 6), mientras que 56.85 se convierte en 56.8.

Como puede verse, a causa de los redondeos, una diferencia de sólo 1/100 entre los valores verdaderos de dos variables puede convertirse en una presentación igual a 1/10, incluso si los cálculos se realizan con todas las cifras.

TEST 16

1 / Una **matriz** es una estructura matemática que puede representarse con una matriz de dos dimensiones (filas y columnas). Por ejemplo, una matriz de 4 × 5 (4 filas y 5 columnas) puede representarse con la matriz A (4,5).
Escribir un programa que lea de la pantalla todos los elementos de la primera fila de la matriz A [los elementos de la primera fila son cinco: A(1,1), A(1,2), ..., A(1,5)] y calcule los otros elementos como producto del número de filas de cada uno de ellos por el correspondiente elemento de la fila 1. [Por ejemplo, para la fila 2 se tiene A(2,1) = 2 * A(1,1), etc.; para la fila 3 A(3,1) = 3 * A(1,1) etc.].

2 / Escribir una subrutina que imprima la matriz A(4,5) preparada en la respuesta al punto anterior, bien en modo normal (fila por fila), bien invertida (columna por columna).

3 / Utilizando la función INPUT$(N), escribir una subrutina que lea (con eco) una sigla formada por 3 caracteres, comprobando que no se introduzcan valores numéricos.

4 / Generalizar la rutina anterior adaptándola a la lectura de un número N de caracteres comprendidos entre 1 y 40.

Las soluciones, en la pág. 567.

Estructuración de las impresiones

Las subrutinas de impresión pueden estructurarse de dos modos principales: escribiendo las instrucciones según un formato específico o parametrizando la fila que se envía a impresión. El primer método es el más usual, y consiste en escribir para cada aplicación específica una rutina dedicada que contiene las instrucciones necesarias, que en ningún caso resultará adaptable a otros programas. El segundo método, mucho más complejo, consiste en preparar una variable de cadena que contiene todas las informaciones que se desea tener en una línea de impresión. La rutina de impresión se reduce a la emisión de esta variable, que toma el nombre de **vector de impresión**. El primer método es rígido: si se tiene que variar uno de los parámetros, es necesaria la ayuda del programador, ya que es preciso modificar las instrucciones. El segundo puede ser mucho más generalizado, hasta el punto de obtener un programa que indica al usuario cuáles son las variables que debe imprimir y en qué posición sobre la hoja.
A continuación se incluye un ejemplo informativo a fin de presentar una metodología cuyo conocimiento, aunque sea superficial, es útil para la comprensión de los paquetes dedicados (Visicalc, Multiplan, Manager, etc.) que pueden implantarse en la mayor parte de los microordenadores y ordenadores personales.

Generación de listados no parametrizados. Este primer ejemplo se refiere al modo usual de programar, que consiste en escribir las instrucciones adecuadas para resolver el problema particular y sólo éste. Se debe imprimir un registro de clientes que contiene las informaciones indicadas en la tabla 1 y los datos económicos indicados en la tabla 2 (ver la página de enfrente). La impresión debe prever el salto de página al cambiar el cliente (ruptura del código de cliente) o cada 30 líneas de datos económicos, incluso si el cliente es el mismo. Las páginas deben estar numeradas y, al final del output relativo a cada cliente, se desea el total del importe neto (suma de los campos Z2).
El primer paso consiste en la preparación del esquema de impresión. Debe prepararse una hoja cuadriculada (existen hojas preparadas) sobre la cual se indicarán las posibles columnas de impresión, las filas de la parte de los datos personales y las 30 filas (máximo previsto por página) de los datos económicos. Sobre esta hoja, después de haberla dividido en coordenadas (filas y columnas), deberán indicarse los campos a imprimir en las posiciones deseadas.

De esta manera, la posición de cada campo queda determinada por las coordenadas de fila y columna. La escritura del programa se reduce a traducir en instrucciones todo lo que se ha representado en la hoja. En la página 562 se ha indicado un ejemplo de disposición de los campos en 80 columnas; la generalización a 132 columnas es inmediata y puede obtenerse distanciando más los campos. En la parte superior de la página 563 se ha representado el diagrama de flujo del main, y en la parte de abajo, el diagrama de flujo de la subrutina de impresión. Las funciones que esta última realiza son:

1 / Reconocimiento del código de cliente (instrucción 1010); a la ruptura del código debe imprimir el total (Z3) que se refiere al cliente anterior, poner a 0 la variable para acumularle los nuevos datos e imprimir los datos del nuevo cliente.

2 / Reconocimiento de la primera página (instrucción 1030); la lógica anterior no puede aplicarse al primer cliente que se imprime, ya que no existen datos anteriores (en otras palabras, Z3 no debe imprimirse). El reconocimiento se obtiene controlando el número de la página: si este número es 0, se imprime el encabezado saltando el total anterior.

3 / Control de las filas (instrucciones 1340, 1380); con cada línea de detalle impresa se incrementa un contador. Cuando se llega al valor 31 debe pasarse a una nueva página, incluso si el cliente no ha cambiado.

En el diagrama de flujo de la página 563 (abajo) puede verse que la impresión de la cabecera y de los datos está prevista en dos puntos diferentes del programa. En estos casos es conveniente utilizar una subrutina en lugar de volver a escribir las instrucciones. Así se tienen dos ventajas: no se repiten las instrucciones y se separan limpiamente las diversas funciones, haciendo más sencilla la comprensión del programa y los eventuales cambios para adaptarlo a nuevas necesidades. En la página 564 se ha representado el diagrama de flujo de la subrutina de impresión de la cabecera y de los datos del cliente. Esta subrutina se lanza a continuación de la ruptura de código o después de la impresión de 30 líneas de detalle. En la página 564 se ha incluido el listado del programa, y en la página 566 algunas salidas. En este ejemplo, la introducción de los datos está prevista por teclado en el main con el solo objeto de mostrar el funcionamiento de las rutinas de impresión.

Tabla 1

Nombre del campo	Significado	Tipo	Longitud
CD	Código de cliente	numérico	3
AP$	Apellido	alfabético	15
CA$	Calle	alfabético	10
NU$	Número	alfabético	5
TL$	Teléfono	alfabético	12 (prefijo y número)
DP$	Distrito postal	alfabético	5
CT$	Ciudad	alfabético	15
PV$	Provincia	alfabético	5

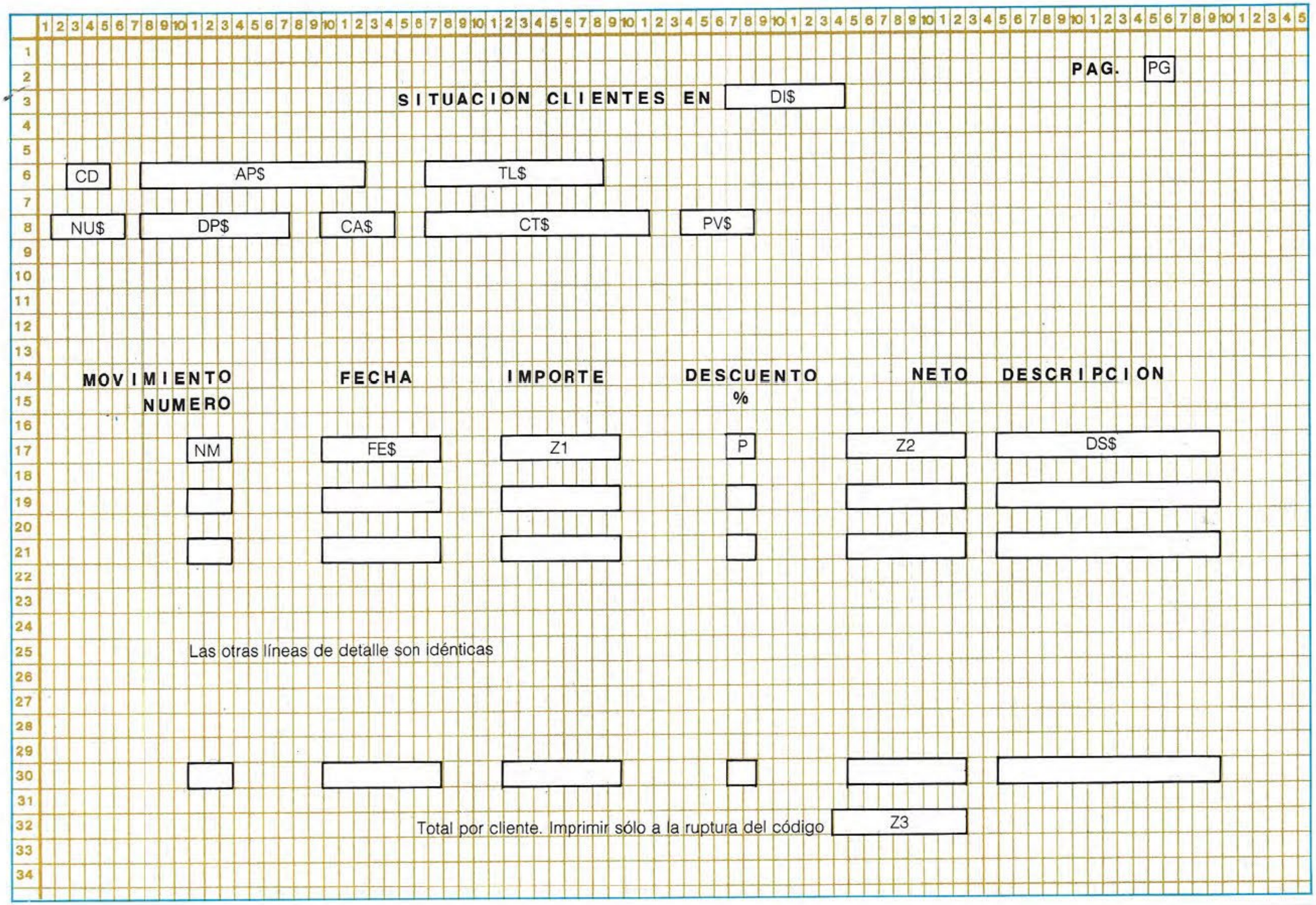
Tabla 2

Nombre del campo	Significado	Tipo	Longitud
NM	Número del movimiento	numérico	3
FE$	Fecha	alfabético	8
Z1	Importe	numérico	8
P	Descuento (%)	numérico	2
Z2	Neto	numérico	8
DS$	Descripción	alfabético	15

Impresora bidireccional PR1350. Un microprocesador controla las posibilidades de escritura.



Olivetti


PAG. PG
SITUACION CLIENTES EN DI$
CD
AP$
TL$
NU$
DP$
CA$
CT$
PV$
MOVIMIENTO NUMERO
FECHA
IMPORTE
DESCUENTO %
NETO
DESCRIPCION
NM
FE$
Z1
P
Z2
DS$
Las otras líneas de detalle son idénticas
Total por cliente. Imprimir sólo a la ruptura del código
Z3

PROGRAMA DE IMPRESION DE TABULADOS DE 80 COLUMNAS



DIAGRAMA DE FLUJO DE LA SUBRUTINA DE IMPRESION DEL REGISTRO DE CLIENTES

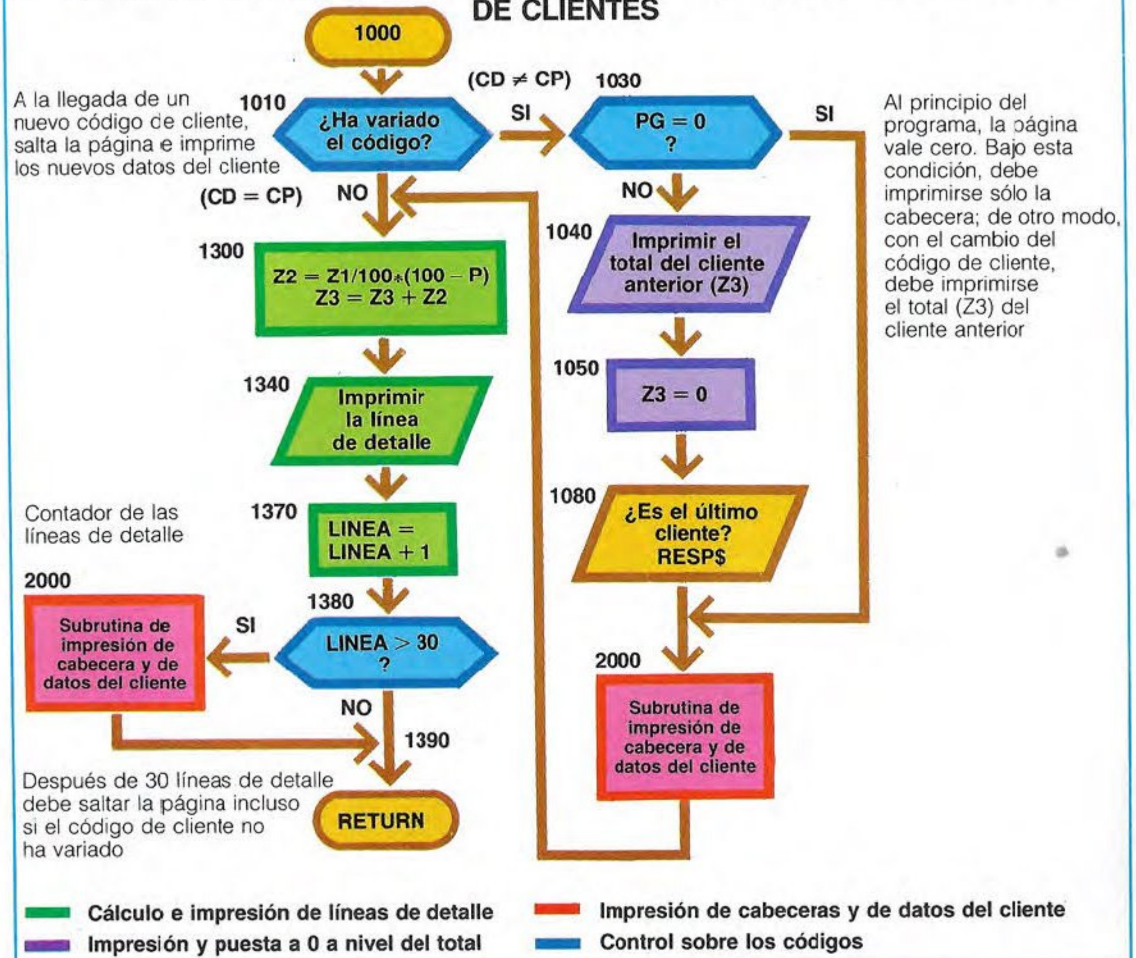
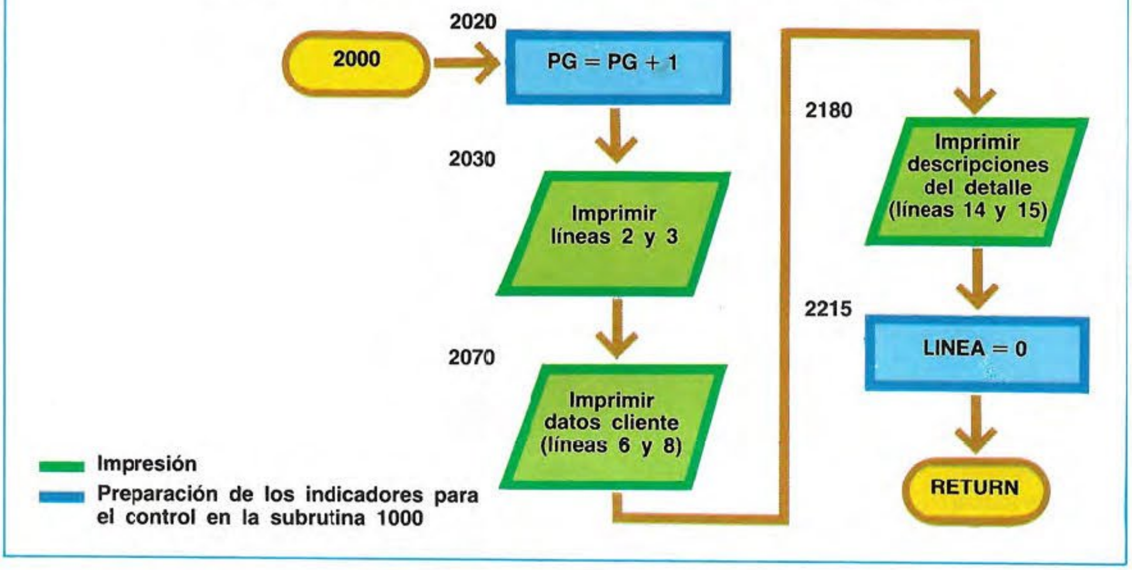


DIAGRAMA DE FLUJO DE LOS DATOS DEL CLIENTE



PROGRAMA PARA TABULADOS DE 80 COLUMNAS

```
5 ' ** PROGRAMA PARA TABULADOS DE 80 COLUMNAS
10 '
15 '              FILE = TAB80C
20 '
25 ' ASIGNACIONES ***********
30 '
35 ' NOMBRE DEL CAMPO      SIGNIFICADO                     TIPO        LONGITUD
40 ' CD                   CODIGO DEL CLIENTE              NUMERICO       3
45 ' AP$                  APELLIDO                        ALFABETICO    15
50 ' CA$                  CALLE                           ALFABETICO    10
55 ' NU$                  NUMERO                          ALFABETICO     5
60 ' TL$                  TELEFONO                        ALFABETICO    12
65 ' DP$                  DISTRITO POSTAL                 ALFABETICO     5
70 ' CT$                  CIUDAD                          ALFABETICO    15
75 ' PV$                  PROVINCIA                       ALFABETICO     5
80 '     ***DATOS ECONOMICOS ***
85 ' NM                   NUMERO DEL MOVIMIENTO           NUMERICO       3
90 ' FE$                  FECHA                           ALFABETICO     6
95 ' Z1                   IMPORTE                         NUMERICO       8
100 'P                    DESCUENTO (%)                   NUMERICO       2
105 'Z2                   NETO                            NUMERICO       8
110 'DS$                  DESCRIPCION                     ALFABETICO    15
115 '
120 ' CD = CODIGO ACTUAL
125 ' CP = CODIGO ANTERIOR
130 '
135 '
140 BI$=CHR$(27)+"+"+CHR$(7)        'LIMPIA LA PANTALLA Y EMITE UN BIP
145 '          ************
146 INPUT "DATOS DE LA IMPRESORA"; DI$
150 CP=0     ' INICIALIZA LA VARIANTE CP CON EL VALOR CERO AL PRINCIPIO
152 '       LECTURA DE LOS DATOS
155 PRINT " INSERTAR EL CODIGO DEL CLIENTE (NUMERICO, MAX, 3 CIFRAS)"
160 INPUNT CD: IF  CD=CP  GOTO 168      Asigna el cod. cliente a la variable CD
161 INPUT "APELLIDO "; AP$
162 INPUT "CALLE "; CA$
163 INPUT "NUMERO "; NU$
164 INPUT "TELEFONO "; TL$
165 INPUT "CIUDAD "; CT$
166 INPUT "DISTRITO POSTAL " ; DP$
167 INPUT "PROVINCIA "; PV$
```

```
168 INPUT "NUMERO DEL MOVIMIENTO "; NM
169 INPUT "FECHA "; FE$
170 INPUT "IMPORTE ." ; Z1
171 INPUT "DESCUENTO (%)" ;  P
172 INPUT "DESCRIPCION : " ,DS$
175 GOSUB 1000  ' (CALCULOS E IMPRESION)
180 CP=CD
185 '
190 PRINT BI$
195 IF RESP$ <> "SI" GOTO 155
200 END
1000 '   ** SUBRUTINAS DE IMPRESION REGISTRO CLIENTES **
1010 ' IF CD=CP GOTO 1300      ' La situación CD=CP indica que no ha
1020 '                          habido variación de código
1030 IF PG=0   THEN GOSUB 2000:GOTO 1300
1040 LPRINT TAB(55);:LPRINT USING "ffffffff"; Z3
1050 Z3=0
1060 LPRINT CHR$(12)              ' INSTRUCCION PARA EL SALTO DE PAGINA
1070 PRINT "¿ES EL ULTIMO CLIENTE? (SI/NO)"
1080 INPUT RESP$
1100 GOSUB 2000         ' (IMPRESION DATOS CLIENTE)
1110 GOTO 1300
1120 '
1130 '
1140 '
1150 '
1300 Z2=Z1/100*(100-P)  ' Restar el descuento del importe hallando así el
1310 '                    NETO, que se memoriza en la variable Z2
1320 Z3=Z3+Z2           ' Acumula en la variable Z3 los importes NETOS
1330 '
1340 '       IMPRIME LA LINEA DE DETALLE
1350 LPRINT TAB(11);: LPRINT USING "fff"; NM; : LPRINT TAB(20); FE$;
1352 LPRINT TAB(32);: LPRINT USING "fffffffff"; Z1;
1354 LPRINT TAB(47);: LPRINT USING "ff"; P;
1356 LPRINT TAB(55);: LPRINT USING "ffffffff"; Z2;:LPRINT TAB (65); DS$
1360 LPRINT
1370 LINEA=LINEA+1
1380 IF LINEA>30 THEN LPRINT CHR$(12):GOSUB 2000
1390 RETURN
1400 '
2000 ' ** SUBRUTINA DE IMPRESION DE CABECERA Y DATOS CLIENTE **
2010 '
2020 PG=PG+1
2030 LPINT TAB(70); "PAG."; PG
2040 LPRINT TAB(25); "SITUACION DEL CLIENTE EL", TAB(47); DI$
2050 LPRINT:LPRINT
2060 '
2070 ' IMPRIME DATOS CLIENTE
2080 '
2090 LPRINT TAB(3); CD; TAB(8); AP$; TAB(27); TL$
2100 LPRINT
2110 LPRINT TAB(2); NU$; TAB(8); CA$; TAB(20); DP$; TAB(27); CT$; TAB(44); PV$
2120 LPRINT
2130 LPRINT
2140 LPRINT
2150 LPRINT
2160 LPRINT
2170 '
2180 LPRINT TAB(5); "MOVIMIENTO"; TAB(20); "IN FECHA"; TAB(32); "IMPORTE";
2190 LPRINT TAB(45); "DESCUENTO"; TAB(58); "NETO"; TAB(65); "DESCRIPCION"
2200 LPRINT TAB(8); "NUMERO"; TAB (47); " % "
2210 LPRINT
2215 LINEA=0
2220 RETURN
```

SALIDAS DEL PROGRAMA TAB. 80C (PAG. 1)

```
                                                            PAG. 1
                    SITUACION CLIENTES EL 4/01/83

  1   PEREZ                (06)258810

123/B  CALLE DUERO    00171    MADRID              MA


MOVIMIENTO   IN FECHA         IMPORTE     DESCUENTO     NETO     DESCRIPCION
  NUMERO                                      %
       936   10/10/83         123000        10        110700   MERCADERIAS FRAGILES
       937   11/10/83         145000        15        123250   MERCADERIAS VARIAS
       938   12/10/83         900000        18        738700   EXPLOSIVO
       939   13/10/83       11920000        10      10728000   MEDICINAS
                                                    11700000
```

SALIDAS DEL PROGRAMA TAB. 80C (PAG. 2)

```
                                                            PAG. 2
                    SITUACION CLIENTES EL 4/01/83

   2   LOPEZ                 (06)484800

  64/A CALLE EBRO      00171 MADRID                  MA


  MOVIMIENTO   IN FECHA      IMPORTE        DESCUENTO      NETO    DESCRIPCION
    NUMERO                                      %
         800   10/11/83      970000             5        921500    JUGUETES
```

Generación de listados parametrizados. La aplicación se desarrolla sólo a nivel informativo y está limitada a un caso muy sencillo, excluyendo el uso de las máscaras de vídeo, que son indispensables en este tipo de programas. Sin embargo, el ejemplo es útil para adquirir familiaridad con algunas instrucciones del lenguaje Basic y para tener una información genérica sobre metodologías particulares.
En la pág. 568 se ha incluido el diagrama de

SOLUCIONES DEL TEST 16

1 / El programa que realiza las funciones deseadas puede asumir la siguiente forma:

```
10 OPTION BASE 1
20 DIM A(4,5)
25 'Input de los elementos de
   la primera línea
30 FOR I = 1 TO 5
40 PRINT "Elemento:"; I
50 INPUT "Valor"; A(1,I)
60 NEXT I
70 'Cálculo de los elementos de
   la segunda línea
80 FOR I =1 TO 5
90 A(2,I) = 2 * A(1,I)
100 NEXT I
110 'Cálculo de los elementos de
    la línea 3
120 FOR I = 1 TO 5
130 A(3,I) = 3 * A(1,I)
140 NEXT I
150 'Cálculo de los elementos de
    la línea 4
160 FOR I = 1 TO 5
170 A(4,I) = 4 *A(1,I)
180 NEXT I
```

Los 3 bucles que calculan los elementos de las tres líneas 2, 3 y 4 pueden reunirse en uno solo. Las instrucciones de la línea 70 a la línea 180 pueden sustituirse por las siguientes:

```
 70 FOR J = 2 TO 4 'Elección de la línea
 80 FOR I = 1 TO 5
 90 A(J,I) = J * A(1,I)
100 NEXT I
110 NEXT J
```

---

2 / La rutina consiste en dos bucles, uno para la elección de la línea y el otro de la columna. Según que índice se haga variar más rápidamente, se puede tener la impresión por filas o por columnas. La forma del bucle es idéntica a la considerada a la solución del punto anterior, sustituyendo las dos líneas siguientes:

```
70 FOR J = 1 TO 4
90 PRINT A(J,I)
```

---

3 / La rutina puede escribirse utilizando tres veces (una por cada carácter) la instrucción A$ = INPUT$(1), a la cual debe seguir el control de numericidad (el valor de A$ debe estar comprendido entre 48 y 57, ver la tabla ASCII). Si el dato introducido no es numérico, en pantalla deberá aparecer A$. En caso contrario, la rutina debe pedir el valor correcto de A$.

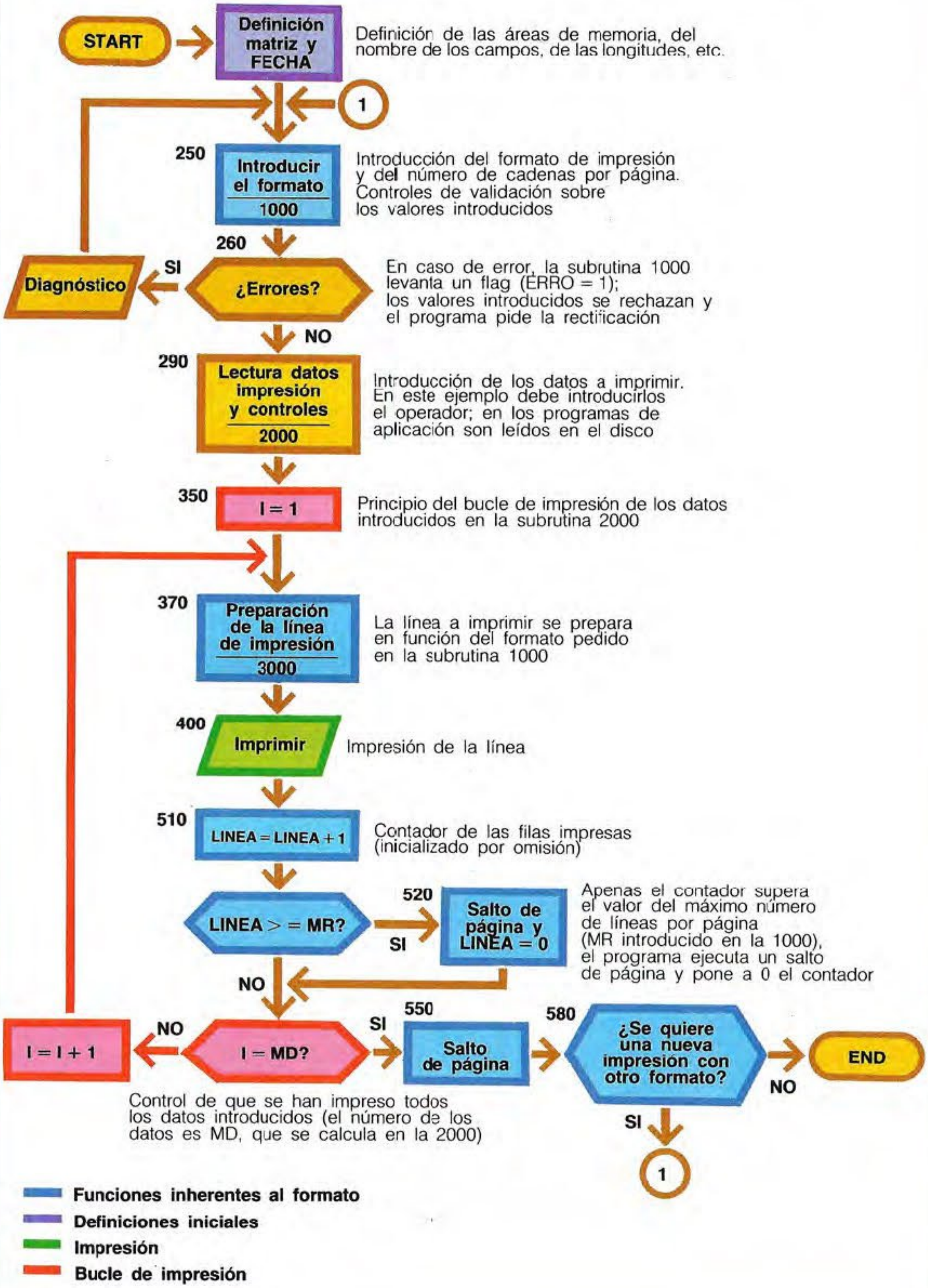
---

4 / La generalización se obtiene utilizando un bucle entre 1 y N que desarrolla las siguientes funciones:

lectura de un carácter [A$ = INPUT$(1)]
control de numericidad
si el dato es aceptado, escritura en pantalla

El programa es muy similar al indicado en la página 555 como ejemplo de uso de la función INPUT$(N).

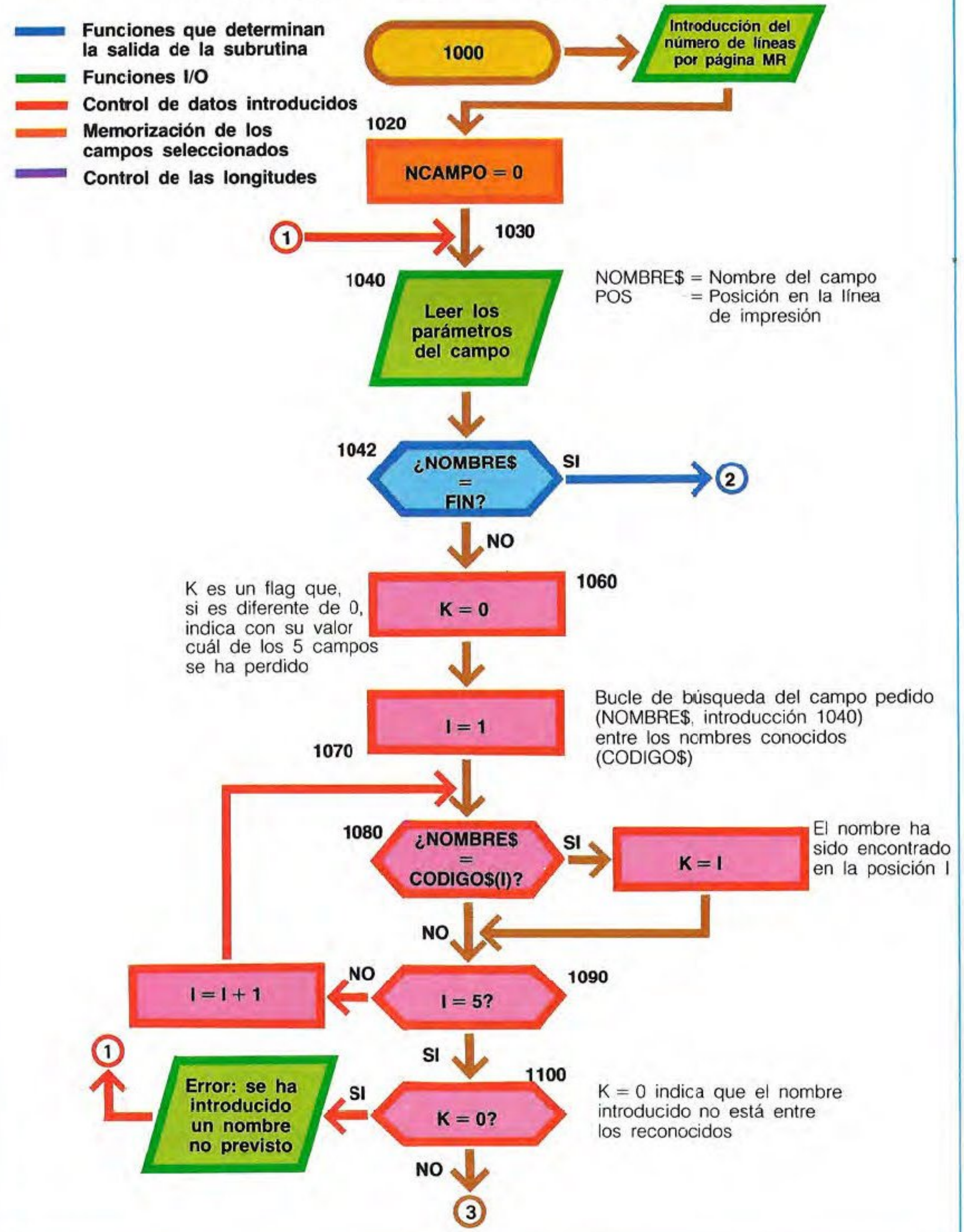
DIAGRAMA DE FLUJO DE IMPRESION PARAMETRIZADA

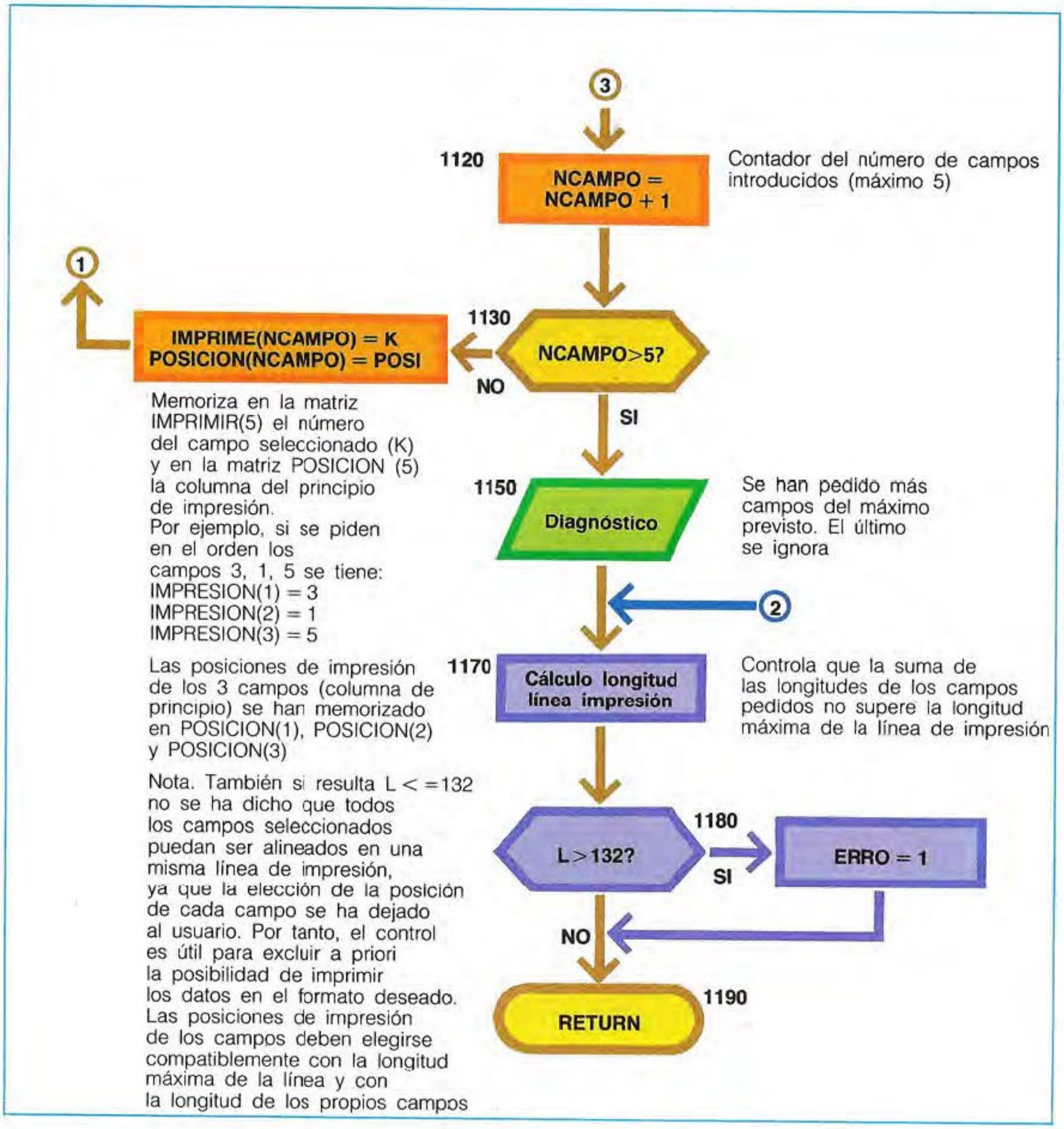


flujo de primer nivel que sintetiza las funciones a desarrollar; en las págs. 569 a 573, los diagramas de flujo de detalle y en las págs. 574 a 576 el listado completo con algunas salidas.

La preparación de una subrutina de impresión parametrizada consiste en definir un cierto número de campos (en el ejemplo 5) cuyos nombres y longitudes son memorizados en el pro-

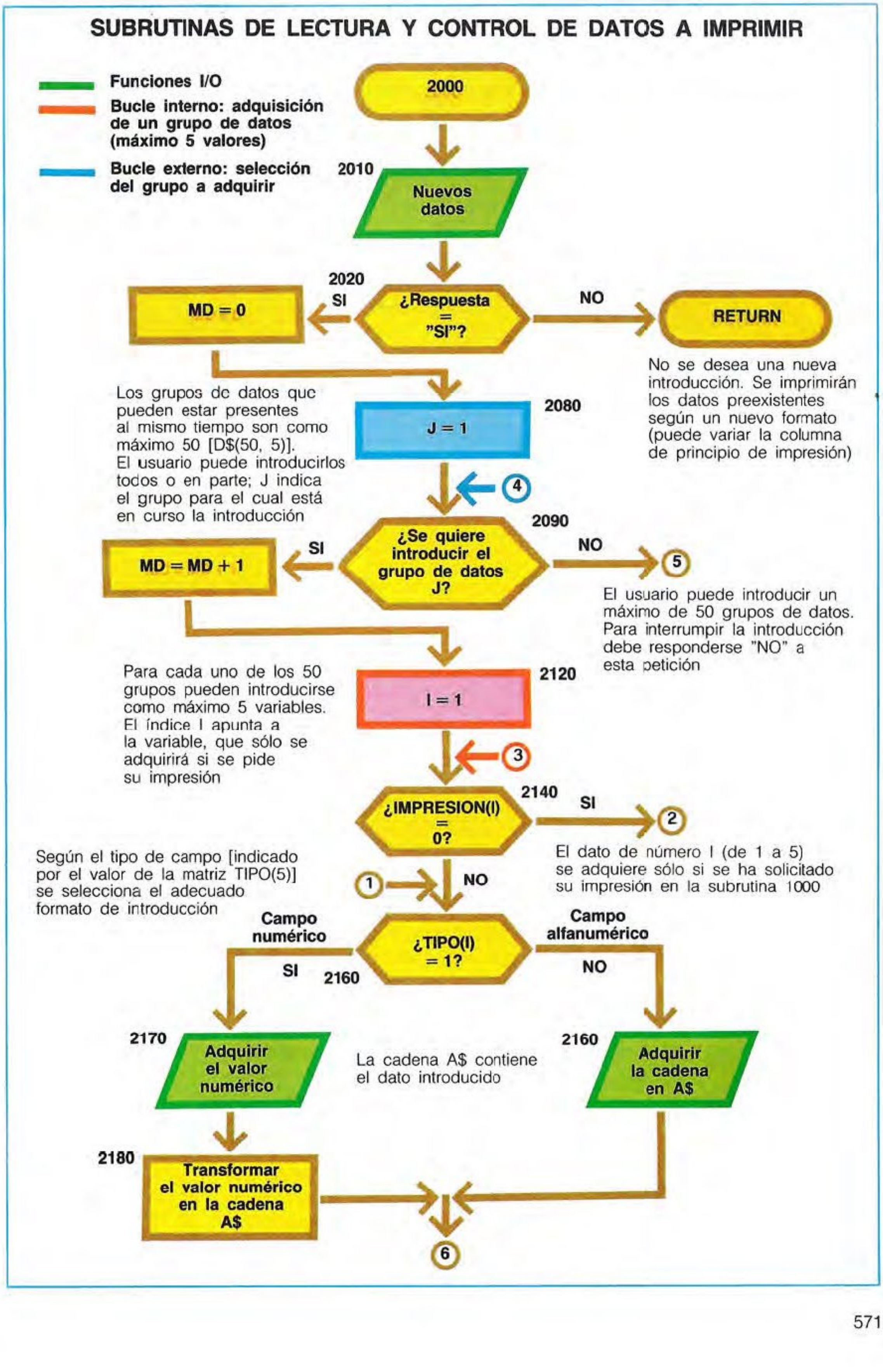
SUBRUTINA DE INTRODUCCION DE FORMATO



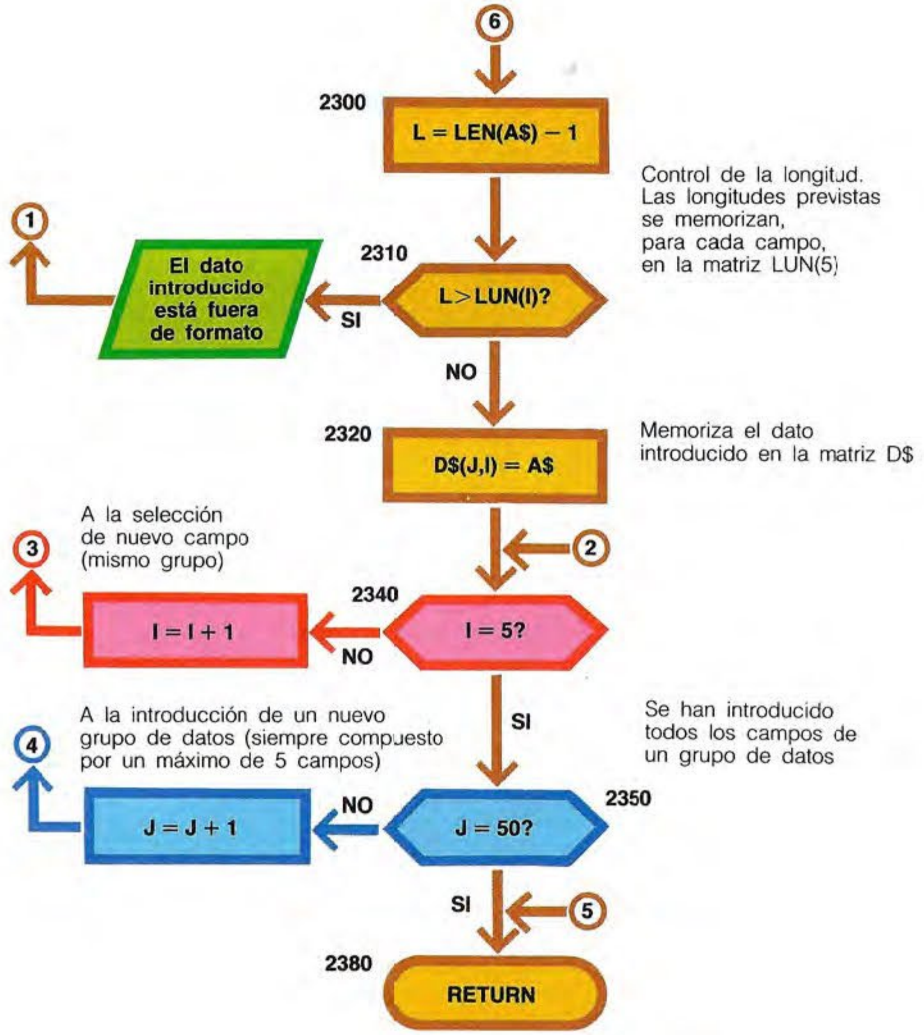


grama (por ejemplo, utilizando la instrucción DATABLE B); el usuario sólo debe especificar qué campos desea elaborar y en qué posición quiere que se impriman. En el ejemplo indicado no se han previsto operaciones aritméticas o lógicas entre los campos, sino la posibilidad de utilizar como datos de entrada valores memorizados en el disco (los datos de entrada son introducidos por teclado).

Completando el software de aplicación con esas funciones y con otras más sofisticadas, pueden obtenerse programas de uso generalizado, gracias a los cuales el usuario final puede desarrollar una discreta cantidad de elaboracio-

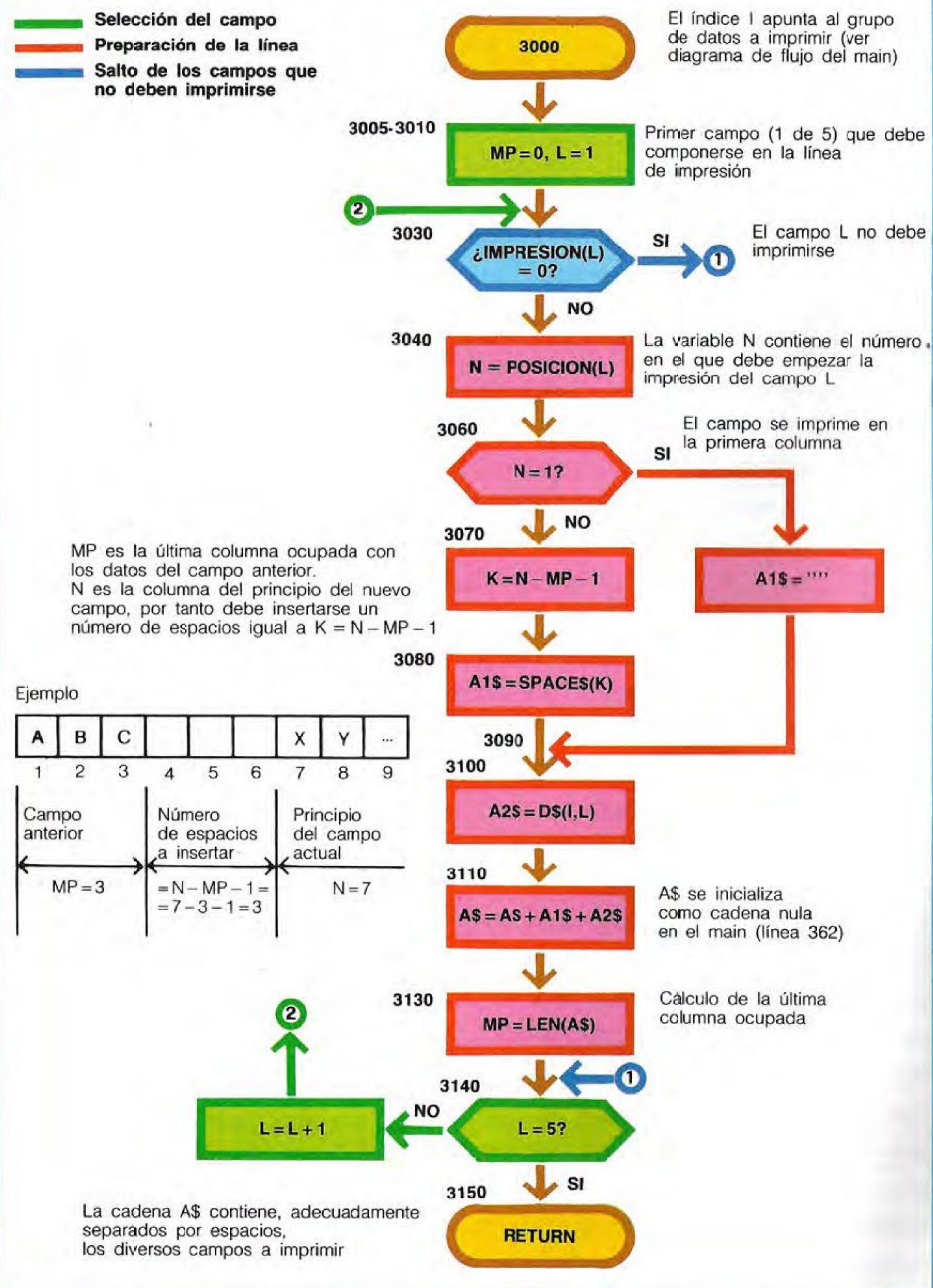


SUBRUTINAS DE LECTURA Y CONTROL DE DATOS A IMPRIMIR
Funciones I/O
Bucle interno: adquisición de un grupo de datos (máximo 5 valores)
Bucle externo: selección del grupo a adquirir
2000
2010
Nuevos datos
2020
SI
¿Respuesta = "SI"?
NO
MD = 0
RETURN
No se desea una nueva introducción. Se imprimirán los datos preexistentes según un nuevo formato (puede variar la columna de principio de impresión)
Los grupos de datos que pueden estar presentes al mismo tiempo son como máximo 50 [D$(50, 5)]. El usuario puede introducirlos todos o en parte; J indica el grupo para el cual está en curso la introducción
2080
J = 1
4
2090
SI
¿Se quiere introducir el grupo de datos J?
NO
MD = MD + 1
5
El usuario puede introducir un máximo de 50 grupos de datos. Para interrumpir la introducción debe responderse "NO" a esta petición
Para cada uno de los 50 grupos pueden introducirse como máximo 5 variables. El índice I apunta a la variable, que sólo se adquirirá si se pide su impresión
2120
I = 1
3
2140
¿IMPRESION(I) = 0?
SI
2
El dato de número I (de 1 a 5) se adquiere sólo si se ha solicitado su impresión en la subrutina 1000
Según el tipo de campo [indicado por el valor de la matriz TIPO(5)] se selecciona el adecuado formato de introducción
1
NO
Campo numérico
SI
¿TIPO(I) = 1?
2160
Campo alfanumérico
NO
2170
Adquirir el valor numérico
La cadena A$ contiene el dato introducido
2160
Adquirir la cadena en A$
2180
Transformar el valor numérico en la cadena A$
6



nes sin entrar a fondo en la programación. Si se está motivado por necesidades de aplicación, el desarrollo de este tipo de software presenta notables dificultades y requiere el uso de lenguajes más cercanos a la máquina de lo que es el Basic. Normalmente, los paquetes de aplicación más complejos y generalizados son escritos en Assembler, cuando no lo son en lenguaje máquina (es decir en forma de códigos hexadecimales); sólo recientemente se está difundiendo el uso del Pascal que, aunque conservando la inmediatez de los lenguajes de alto nivel, permite, dada su particular estructura, desarrollar programas muy complejos con notable ahorro de tiempo con respecto al Assembler.

PREPARACION DE LA LINEA DE IMPRESION



PROGRAMA DE IMPRESION PARAMETRIZADA

```
10 ' ** PROGRAMA DE IMPRESION PARAMETRIZADA **
20 '
30 OPTION BASE 1
40 '
50 '                    FILE = PROPAR
60 ' DECLARACIONES Y ASIGNACIONES
70 '
80 '
82 ' MR = NUMERO MAX. DE LINEAS POR PAGINA (SUB. 1000)
84 ' MD = NUMERO DE GRUPOS DE DATOS QUE SE QUIEREN IMPRIMIR (SUB. 2000)
86 DEFINT I-N
90 BI$=CHR$(27)+"+"+CHR$(7)
100 DIM CODIGOS$(5)         ' CONTIENE LOS 5 NOMBRES DE LOS CAMPOS RECONOCIDOS
110 DIM LON(5)              ' LONGITUD DE LAS 5 VARIABLES
120 DIM TIPO(5)             ' TIPO DE LAS VARIABLES 1=NUMERICO
130 '                         2 = ALFANUMERICO
140 DIM POSICION (5)        'COLUMNA DE INICIO DE IMPRESION DE CADA UNO DE LOS 5
150 '                         CAMPOS POSIBLES
160 DIM IMPRESION (5)       ' CONTIENE, EN EL ORDEN DESEADO, LOS PUNTEROS
170 '                         A LOS CAMPOS A IMPRIMIR
180 DIM D$ (50,5)           ' MATRIZ DE APOYO EN LA QUE SE ENDOSAN
190 '                         LOS DATOS
192 FOR I=1 TO 5
194 PRINT "CAMPO N.  "; I
196 INPUT  "CODIGO (3 CARACTERES)";CODIGOS$(I)
198 INPUT "LONGITUD EN CARACTERES ";LON(I)
200 INPUT "TIPO (1=NUMERICO , 2=ALFANUMERICO) "; TIPO (I)
202 LPRINT "CAMPO N.:  "; I, "CODIGO:"; "CODIGOS$:(I)
204 LPRINT "NUMERO DE CARACTERES: "LON(I)." TIPO: ";TIPO(I)
206 NEXT I
210 '
220 ' SUBRUTINA 1000 = PARA LA INTRODUCCION DEL FORMATO DE IMPRESION
230 ' Y DEL N. DE LINEAS POR PAGINA. CONTROLES DE VALIDEZ SOBRE LOS VALORES
240 ' INTRODUCIDOS
250 GOSUB 1000
260 IF ERRO=1 THEN PRINT BI$:PRINT "ERROR, RECTIFICAR LOS VALORES ";
270 IF ERRO=1 THEN PRINT "INTRODUCIDOS":GOTO 250
280 '
290 ' SUBRUTINA 2000 = PARA LA INTRODUCCION DE LOS DATOS A IMPRIMIR
300 ' EN ESTE EJEMPLO LOS INTRODUCE EL OPERADOR (En los programas
310 ' de aplicación se leen en el DISCO)
320 '
330 GOSUB 2000
340 '
350 FOR I=1 TO MD
360 ' PRINCIPIO DEL BUCLE DE IMPRESION DE LOS DATOS INTRODUCIDOS POR
361 ' LA SUBRUTINA 2000
362 A$=""
370 GOSUB 3000          'Se prepara la línea a imprimir en función
380 '                   del formato pedido por la Subrutina 1000
390 '    INSTRUCCIONES DE IMPRESION DE LA LINEA
400 LPRINT A$
410 '
490 '
500 '      CONTADOR DE LINEAS IMPRESAS
510 LINEA=LINEA+1
520 IF LINEA>=MR THEN LPRINT CHR$(12): LINEA=0
530 NEXT I
540 '
550 LPRINT CHR$(12)        ' INSTRUCCION EN CODIGO PARA SALTO DE PAGINA
560 '
570 PRINT BI$              'LIMPIA LA PANTALLA Y EMITE UN BIP
580 PRINT   "¿SE DESEA UNA IMPRESION CON UN NUEVO FORMATO? (SI/NO)"
590 INPUT RESP$
```

```
600 IF RESP$="SI" THEN GOTO 192
610 '
620 END
630 '
900 '  ******  FIN MAIN  *******
1000 ' **SUBRUTINA PARA LA INTRODUCCION DEL FORMATO  **
1010 '
1012 PRINT BI$
1014 PRINT "INTRODUCIR EL NUMERO DE LINEAS POR PAGINA"
1016 INPUT MR
1020 NCAMPO=0          INICIALIZA LA VARIABLE (NCAMPO) AL VALOR CERO
1030 '
1040 '   LECTURA DE LOS PARAMETROS DEL CAMPO
1041   INPUT "NOMBRE DEL CAMPO "; NOMBRE$
1042 IF NOMBRE$="FIN" GOTO 1160
1044 INPUT "POSICION EN LA LINEA DE IMPRESION "; POSI
1060 K=0   ' FLAG cuyo valor, si es diferente de cero, indica cuál de los
1065 '        5 campos se ha pedido
1070 FOR I=1  TO 5
1075 '  INICIO BUCLE DE BUSQUEDA DEL CAMPO PEDIDO (NOMBRE$, LINEA 1041)
1077 '  ENTRE LOS NOMBRES RECONOCIDOS (CODIGOS$) LINEA 100
1080 IF NOMBRE$=CODIGOS$(I) THEN K=I EL NOMBRE SE HA ENCONTRADO EN
1085 '                                POSICION (I)
1090 NEXT I
1100 IF K=0 THEN PRINT "ERROR, SE HA INTRODUCIDO UN NOMBRE NO PREVISTO":
1101 GOTO 1030
1110 '      K = 0 INDICA QUE EL NOMBRE INTRODUCIDO NO ESTA ENTRE LOS RECONOCIDOS
1120 'NCAMPO=NCAMPO+1
1130 IF NCAMPO <=5 THEN IMPRIME (NCAMPO)=K: POSICION (NCAMPO)=POSI:GOTO 1030
1140 PRINT BI$
1150 PRINT   "SE HAN PEDIDO MAS CAMPOS QUE EL MAX. PREVISTO"
1160 '
1161  L=0
1162 FOR I=1 TO NCAMPO
1163 L=L+LON(IMPRESION (I))
1164  NEXT I
1170 '
1180 IF L>132 THEN ERRO=1
1190 RETURN
2000 ' ** SUBRUTINA DE LECTURA Y CONTROL DE LOS DATOS A IMPRIMIR **
2005 '
2010 INPUT "¿SE DESEA INTRODUCIR NUEVOS DATOS? (SI/NO)"; RESPUESTA$
2020 IF RESPUESTA$="NO" THEN RETURN
2030 PRINT BI$
2040 PRINT "LOS GRUPOS DE DATOS QUE PUEDEN ESTAR PRESENTES AL MISMO TIEMPO"
2050 PRINT "SON UN MAX. DE 50 (D$(50,5)) VER LINEA N. 180"
2060 PRINT "EL USUARIO PUEDE INTRODUCIRLOS TODOS O PARTE"
2070 PRINT
2071 MD=0
2080 FOR J=1 TO 50
2090 PRINT "SE DESEA INTRODUCIR EL GRUPO DE DATOS: ";J;"?";" (SI/NO)"
2100 INPUT RES$
2110 IF RES$="NO" GOTO 2380
2115 MD=MD+1
2120 FOR I=1 TO 5
2130 '
2140 IF IMPRESION (I)=0 GOTO 2340
2142 PRINT"  ENTRADA DE DATOS DEL CAMPO N.: ";I
2150 '
2160 IF TIPO (I) <>1 THEN INPUT "CADENA ";A$: GOTO 2300
2170 INPUT "VALOR "; IV
2180 A$=STR$(IV)
2300 L=LEN(A$)-1
2310 IF L>LON (I) THEN PRINT "EL DATO INTRODUCIDO ESTA FUERA DE FORMATO";
2311 GOTO 2150
2320 D$(J,I)=A$   'MEMORIZA EL DATO INTRODUCIDO EN LA MATRIZ (D$)
```

```
2330 '
2340 NEXT I
2350 NEXT J
2360 '
2380 RETURN
2390 '
2400 '
3000 ' ** SUBRUTINA PARA LA PREPARACION DE LA LINEA DE IMPRESION **
3005 MP=0
3010 FOR L=1 TO 5
3020 '
3030 IF IMPRESION (L)=0 GOTO 3140
3040  N=POSICION    (L)    '  LA VARIABLE (N) CONTIENE EL N. DE COLUMNA
3050 '                     EN LA QUE DEBE EMPEZAR LA IMPRESION DEL CAMPO (L)
3060 IF N=1 THEN A1$="":GOTO 3090
3070 K=N-MP-1
3080 A1$=SPACE$ (K)
3090 '
3100 A2$=D$ (I,L)
3110 A$=A$+A1$+A2$
3120 '  CALCULO DE LA ULTIMA COLUMNA OCUPADA
3130 MP=LEN (A$)
3140 NEXT L
3150 RETURN
```

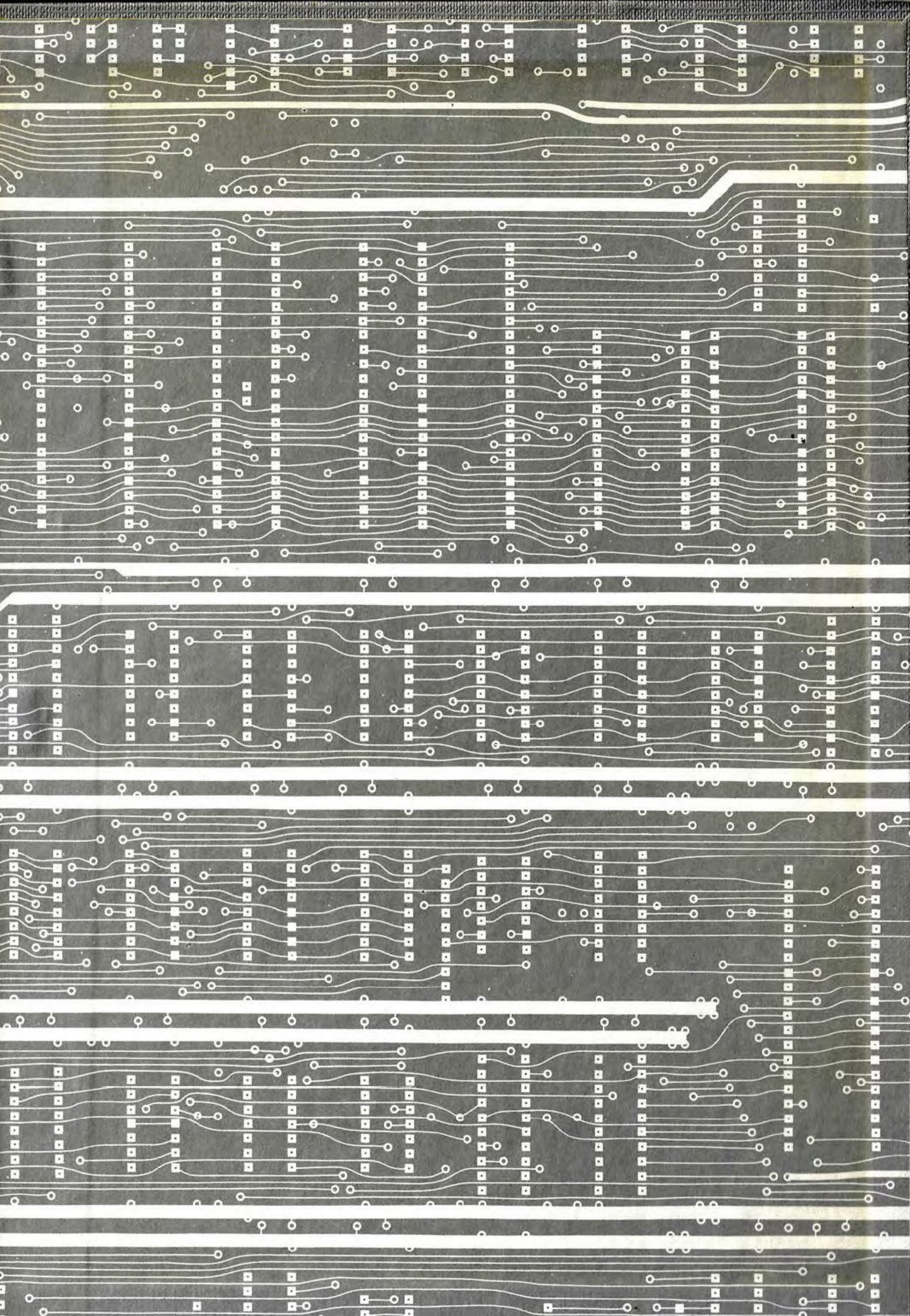
```
CAMPO N.:           1  CODIGO: UNO
NUMERO DE CARACTERES:          3        TIPO:    1
CAMPO N.:           2  CODIGO: DOS
NUMERO DE CARACTERES:          28       TIPO:    2
CAMPO N.:           3  CODIGO: TRE
NUMERO DE CARACTERES:          1        TIPO:    1
CAMPO N.:           4  CODIGO: CUA
NUMERO DE CARACTERES:          1        TIPO:    1
CAMPO N.:           5  CODIGO: CIN
NUMERO DE CARACTERES:          2        TIPO:    1

      123             /XXXXXXXXXXXXXXXXX/                    55
      456             ....................                   66
      888             ABCDEFGHI                              99

CAMPO N.:           1  CODIGO: UNO
NUMERO DE CARACTERES:          3        TIPO:    1
CAMPO N.:           2  CODIGO: DOS
NUMERO DE CARACTERES:          28       TIPO:    2
CAMPO N.:           3  CODIGO: TRE
NUMERO DE CARACTERES:          10       TIPO: 2
CAMPO N.:        4  CODIGO:  CUA
NUMERO DE CARACTERES:          5        TIPO: 2
CAMPO N.:        5  CODIGO:  CIN
NUMERO DE CARACTERES:          2        TIPO: 1

      123   /XXXXXXXXXXXXXXXXX/            55
      456   .........................      66
      888   ABCDEFGHI                      99
```


2330 '
2340 NEXT I
2350 NEXT J
2360 '
2380 RETURN
2390 '
2400 '
3000 ' ** SUBRUTINA
3005 MP=0
3010 FOR L=1 TO 5
3020 '
3030 IF IMPRESION
3040 N=POSICION
3050 '
3060 IF N=1 TH
3070 K=N-MP-1
3080 A1$=SPAC
3090 '
3100 A2$=D$
3110 A$=A$
3120 '
3130 MP=
3140 NEX
3150 RE
CA
N



EDICIONES
FORUM

2

BASIC

ENCICLOPEDIA DE LA INFORMATICA
MINIORDENADORES Y ORDENADORES PERSONALES

